1 2/37
11/21/01
NP

SOFTWARE RELEASE NOTICE

01. SRN Number: **GHGC-SRN-219**

02. Project Title: **Geophysics/Hydrology**

Project No.: **20-1402-861**

03. SRN Title: **FOURIER-FILTER_1D**

04. Originator/Requestor: **David Farrell**

Date: **5/4/200**

05. Summary of Actions

- ■ Release of new software **(Acquired)**
- ☐ Release of modified software:
  - ☐ Enhancements made
  - ☐ Corrections made
- ☐ Change of access software
- ☑ Software Retirement DW 11/30/2001

06. Persons Authorized Access

| Name | Read Only/Read-Write | Addition/Change/Delete |
|---|---|---|
| C. Connor | RO | **Addition** |
| D. Farrell | RW | **Addition** |
| J. Winterle | RO | **Addition** |
| P. La Femina | RO | **Addition** |

07. Element Manager Approval: **Gordon Wittmeyer** Gordon Wittmeyer Date: 5/8/2000

08. Remarks: **None**

CNWRA Form TOP-6 (05/98)

# SOFTWARE SUMMARY FORM

| | | |
|---|---|---|
| 01. Summary Date: 5/3/2000 | 02. Summary prepared by (Name and phone): David Farrell (210) 522-5208 | 03. Summary Action: NEW (Acquired) |
| 04. Software Date: Not Shown | 05. Short Title: FOURIER-FILTER_1D VERSION 1.0 | |
| 06. Software Title: FOURIER-FILTER_1D | | 07. Internal Software |

| | | |
|---|---|---|
| 08. Software Type:<br>☐ Automated Data System<br>■ Computer Program<br>☐ Subroutine/Module | 09. Processing Mode:<br>■ Interactive<br>☐ Batch<br>☐ Combination | 10. Application Area:<br>a. General:<br>■ Scientific/Engineering ☐ Auxiliary Analyses<br>☐ Total System PA<br>☐ Subsystem PA ☐ Other<br>b. Specific:<br>**1D Spectral Analysis** |
| 11. Submitting Organization and Address:<br>CNWRA/SwRI<br>6220 Culebra Road<br>San Antonio, TX 78228 | | 12. Technical Contact(s) and Phone:<br>**David Farrell (210) 522-5208** |

13. Software Application: **The software can be used to perform spectral analysis of 1D time or space data. The software may also be used to filter 1D time or space data. Filters supported include: high-pass filter, low-pass filter, band-pass filter, band-reject filter and notch filter.**

| | | | |
|---|---|---|---|
| 14. Computer Platform: **All** | 15. Computer Operating System: **All** | 16. Programming Language(s): **FORTRAN 77** | 17. Number of Source Program Statements: **1,502 Lines of code** |
| 18. Computer Memory Requirements: **8 Mb** | 19. Tape Drives: **CD-ROM** | 20. Disk Units: **N/A** | 21. Graphics: **N/A** |

22. Other Operational Requirements: **None**

| | |
|---|---|
| 23. Software Availability:<br>☐ Available ☐ Limited ■ In-House ONLY | 24. Documentation Availability:<br>☐ Available ☐ Preliminary ■ In-House ONLY |

25.

Software Developer: David A. Farrell David A. Farrell Date: 5/4/2000

CNWRA Form TOP-4-1 (05/98)

**Software Title:** Fourier-filter_1D (Version 1.0)

**Status:** The software was originally written by David Farrell, Dept. of Geological Sciences, University of Manitoba, Winnipeg, Manitoba, Canada. The software should be considered as acquired so an SRD is not required.

**Author:**

David A. Farrell
CNWRA, SwRI
6220 Culebra Rd.
San Antonio, TX 78238
Tel: 210-522-5208
Email: dfarrell@swri.edu

**Function:** The software is designed for spectral analysis and filtering of one-dimensional spatial (e.g., magnetic and gravity) and temporal (e.g., seismic) datasets. Several types of filtering are supported. These include low-pass filters, high-pass filters, band-pass filters, band-reject filters, and notch filtering. All filtering is performed in the frequency or wave-number domain.

The software also supports a number of popular windowing functions. The currently supported windowing functions include the rectangular window, the tapered rectangular window, the Hamming, the Welch window, and the Bartlett window. Additional windowing functions can be easily incorporated into the program. As a result, the program may be used to develop and test windowing applications.

Convolution and de-convolution capabilities will be added to the software in the near future. This will be done in accordance with the guidelines outline in CNWRA's SCR.

**Language:** The program is written in ANSI Fortran 77. Because there are no proprietary functions or subroutines in the program it should compile using any commercially and publically available Fortran 77 or higher compiler.

**Operating Systems:** The program should compile and run on all available operating systems.

## Installation and Acceptance Tests

To test the installation of the code several test simulations were performed. The following summarizes these test simulations.

The input wave form used in the simulations consisted of three cosine functions summed together (equation 1).

$$y = \sum_{i=1}^{N} A_i \cos(2\pi f_i t + \phi_i)$$

Here $A, f$ and $\varphi$ represent the amplitude, frequency and phase respectively of the $i^{th}$ cosine function, and N represents the number of functions used in the summation. Cosine functions are used for the demonstration because the spectrum of a cosine function is simply a Delta function located at the signal frequency, with magnitude equivalent to the amplitude of the signal. The amplitudes, frequencies and phases of these functions used in the demonstration are:

| | | | |
|---|---|---|---|
| 1. | $A$ = 1.0 m | $f$ = 25 Hz | $\varphi$ = 1.0 rad. |
| 2. | $A$ = 0.4 m | $f$ = 40 Hz | $\varphi$ = 0.0 rad. |
| 3. | $A$ = 1.3 m | $f$ = 55 Hz | $\varphi$ = -0.75 rad. |

These individual time-series along with their aggregate signal are shown in figure 1.

Windowing functions are often used to reduce the spurious effects associated with sharp truncations of signals. The following demonstrates the windowing functions and their impacts on spectral analysis. It should be noted that truncation, that is multiplication, of the signal in the time domain or space domain with the windowing function represents convolution of the spectral responses of the signal and the windowing function in the frequency or wave-number domain.

### Rectangular Window

The rectangular window represents the window in which data is collected, that is, it represents a window that is equal to zero except during the period of sampling when it is equal to unity. Hence the window is characterized by a sharp discontinuity. The window is defined mathematically as

$$w(t) = \begin{cases} 0 & t < T_1 \\ 1 & T_1 \le t \le T_2 \\ 0 & t > T_2 \end{cases}$$

where $w(t)$ represents the window function, and $T_1$ and $T_2$ represents the start and end times of the

window respectively. The signal, the window, and the truncated or windowed function are shown in figure 1. The spectrum of the truncated signal, figure 1, shows energy spread (or leakage) about the characteristic frequencies of the input signal. This leakage is due to the convolution of the energy spectrum of the signal (a series of delta functions in the frequency domain) with the energy spectrum of the rectangular window which contains non-zero frequencies associated with the sharp discontinuity (recall that the spectrum of a constant or DC signal has all of its energy concentrated at zero frequency). Depending on the complexity of the input signal this leakage can complicate spectral analysis by masking characteristic frequencies. Several windowing functions have been designed which produce less energy leakage than the rectangular window.

**Tapered Rectangular Window**

The tapered rectangular window is defined mathematically as

$$w_{tr}(t) = \begin{cases} \frac{1}{2}\left\{1+\cos\left[\pi\left(\frac{t+T_1}{T_1}\right)\right]\right\} & 0 \le t < T_1 \\ 1 & T_1 \le t \le T_2 \\ \frac{1}{2}\left\{1+\cos\left[\pi\left(\frac{t-T_2}{T_3-T_2}\right)\right]\right\} & T_2 < t \le T_3 \end{cases}$$

where $w(t)$ represents the window function, $T_1 = T_3 - T_2$, represents the taper time for the window, and $T_3$ represents the total window length. The length of the taper, $T_1$, is currently hardwired into the code. This constraint will be relaxed in future versions of the code. The tapered rectangular window, the input signal, and the windowed input function are shown in figure 2. The spectrum of the truncated signal is shown in figure 2. The spectrum appears similar to that of the rectangular window. However, the amount of energy leaked into frequencies near the characteristic frequencies of the signal are reduced. In addition, because of the tapering energies at the characteristic frequencies of the signal appear to be marginally reduced.

**Welch Window**

The Welch window is defined mathematically as

$$w(t) = 1 - \left(\frac{t-0.5T}{0.5T}\right)^2 \qquad 0 \le t \le T$$

where $T$ represents the width of the window. The Welch window, the input signal, and the windowed input function are shown in figure 3. The spectrum of the truncated signal is shown in figure 3. The spectrum displays the characteristic frequencies of the input signal. Energy leakage into frequencies surrounding the characteristic frequencies are reduced compared to the input rectangular window. In

addition, the window results in a reduction in the energy at the characteristic frequency.

**Bartlett Window**

Mathematically, the Bartlett window is defined similarly to the Welch window, with the difference being the exponent term. The mathematical definition for the Welch window is

$$w(t) = 1 - \left(\frac{t - 0.5T}{0.5T}\right) \qquad 0 \le t \le T$$

where $T$ represents the width of the window. The Bartlett window, the input signal, and the windowed input function are shown in figure 4. The spectrum of the truncated signal is shown in figure 4. The spectrum displays the characteristic frequencies of the input signal. Energy leakage into frequencies surrounding the characteristic frequencies are reduced compared to the input rectangular window. In addition, the window results in a reduction in the energy at the characteristic frequency.

**Hamming Window**

The Hamming window is defined mathematically as

$$w(t) = \alpha - (1 - \alpha) * \cos\left(2\pi \frac{t}{T}\right) \qquad 0 \le t \le T$$

where $\alpha$ lies in the range 0–1 and $T$ represents the time length of the window. In the program, $\alpha$ is set to its optimal value of 0.54. The Hamming window, the input signal, and the windowed input function are shown in figure 5. The spectrum of the truncated signal is shown in figure 5. The spectrum displays the characteristic frequencies of the input signal. Energy leakage into frequencies surrounding the characteristic frequencies are reduced compared to the input rectangular window. However, the energies at characteristic frequencies are significantly reduced.

**Digitial Filtering**

Several digital filters have been implemented in the current version of the software. These include

- the low-pass filter
- the high-pass filter
- the band-pass filter
- the band-remove or band-reject filter
- the notch filter

All filtering is implemented in the frequency domain. The first four filters are implemented in the frequency domain using Butterworth filters. Mathematical definitions of these filters are provided in the following summaries.

Butterworth Filters

Extensive discussions of Butterworth filters are provided in Kanasewich (1985) and are not reproduced

here.

Butterworth Low-Pass Filter

The mathematical definition of the Butterworth low-pass in the frequency or wave-number domain is

$$\left|Y_L(W)\right|^2 = \frac{1}{1+W^{2n}}$$

where $W=f/f_c$ is the normalized frequency, $f_c$ is the cut-off frequency, and $n$ controls the rate of attenuation around the cut-off frequency. Expressions for the low-pass filter in the time or space domain though available have not been implemented in this version of the software. Figure 6 shows an example of the low-pass Butterworth filter. Figure 6 shows the low-pass filtered input time-series. Here, the cut-off frequency $f_c$ is set to 30 Hz and is designed to pass only the 25 Hz frequency component of the input signal.

Butterworth High-Pass Filter

The mathematical definition of the Butterworth high-pass filter in the frequency or wave-number domain is derived from the frequency or wave-number domain expression for the Butterworth low-pass filter. That is

$$\left|Y_H(W)\right|^2 = 1-\left|Y_L(W)\right|^2 = \frac{1}{1+\left(\frac{1}{W}\right)^{2n}}$$

where the definitions of $W$ are as defined above. Expressions for the high-pass filter in the time or space domain though available have not been implemented in this version of the software. Figure 7 shows an example of the high-pass Butterworth filter. Figure 7 shows the high-pass filtered input time-series. Here, the cut-off frequency $f_c$ is set to 30 Hz and is designed to remove the 25 Hz frequency component of the input signal.

Butterworth Band-Pass Filter

The Butterworth band-pass filter as implemented in this code is based on the combination of Butterworth high-pass and low-pass filters in the frequency or wave-number domain. The approach involves first filtering the data with the low-pass filter and then filtering the resulting data with the high-pass filter. Under this formulation, the low-pass cut-off frequency is higher than the high-pass cut-off frequency. In the frequency or wave-number domain, the band-pass filter may be viewed as the multiplication of the high-pass and low-pass filters. Expressions for the band-pass filter in the time or space domain though available have not been implemented in this version of the software. Figure 8 shows an example of the band-pass Butterworth filter in the frequency or wave-number domain. Figure 8 shows the band-pass filtered input time-series. Here, the upper cut-off frequency $f_{uc}$ is set to 32 Hz and the lower cut-off frequency $f_{lc}$ is set to 18 Hz. Improved versions of this filter will be incorporated

in future versions of this software.

Butterworth Band-Reject or Band-Remove Filter

The Butterworth band-remove or band-reject filter as implemented in this code is identical to that used in the Butterworth band-pass filter. That is, the Butterworth band-pass filter is first constructed in the frequency or wave-number domain and then subtracted from unity to form the band-reject filter. Expressions for the band-pass filter in the time or space domain though available have not been implemented in this version of the software. Figure 9 shows an example of the band-reject Butterworth filter in the frequency. Figure 9 shows the band-reject filtered input time-series. Here, the upper cut-off frequency $f_{uc}$ is set to 32 Hz and the lower cut-off frequency $f_{lc}$ is set to 18 Hz. Improved versions of this filter will be incorporated in future versions of this software.

Notch Filter

The notch filter is designed to remove a specified frequency from a data set. An example of the application of a notch filter is the removal of the 60 Hz frequency component (associated with electric transmission lines) from seismic and electromagnetic data. The notch filter used in this code is based on recursive filtering, and is mathematically defined as

$$Y_{nf}(f) = \frac{(\omega^2 - \omega_0^2)}{(\omega - i\varepsilon\omega_0)^2 - \omega_0^2}$$

where $\omega=2\pi f$, $f$ corresponding to frequency, $\omega_0$ corresponds to the frequency, $f_0$, being filtered from the input data, $\varepsilon$ controls the steepness of the filter about the frequency being removed (currently set $10^{-5}$), and $i^2=-1$ giving the filter both real and imaginary components. Figure 10 shows an example of the notch filter in the frequency domain with the frequency being removed, $f_0$, being set to approximately 25 Hz. Note that the filtered frequency is approximate because the sampling rate being used does not allow the 25 Hz frequency to be sampled. The filtered time series is shown in figure 10. Improved versions of this filter will be incorporated in future versions of this software.

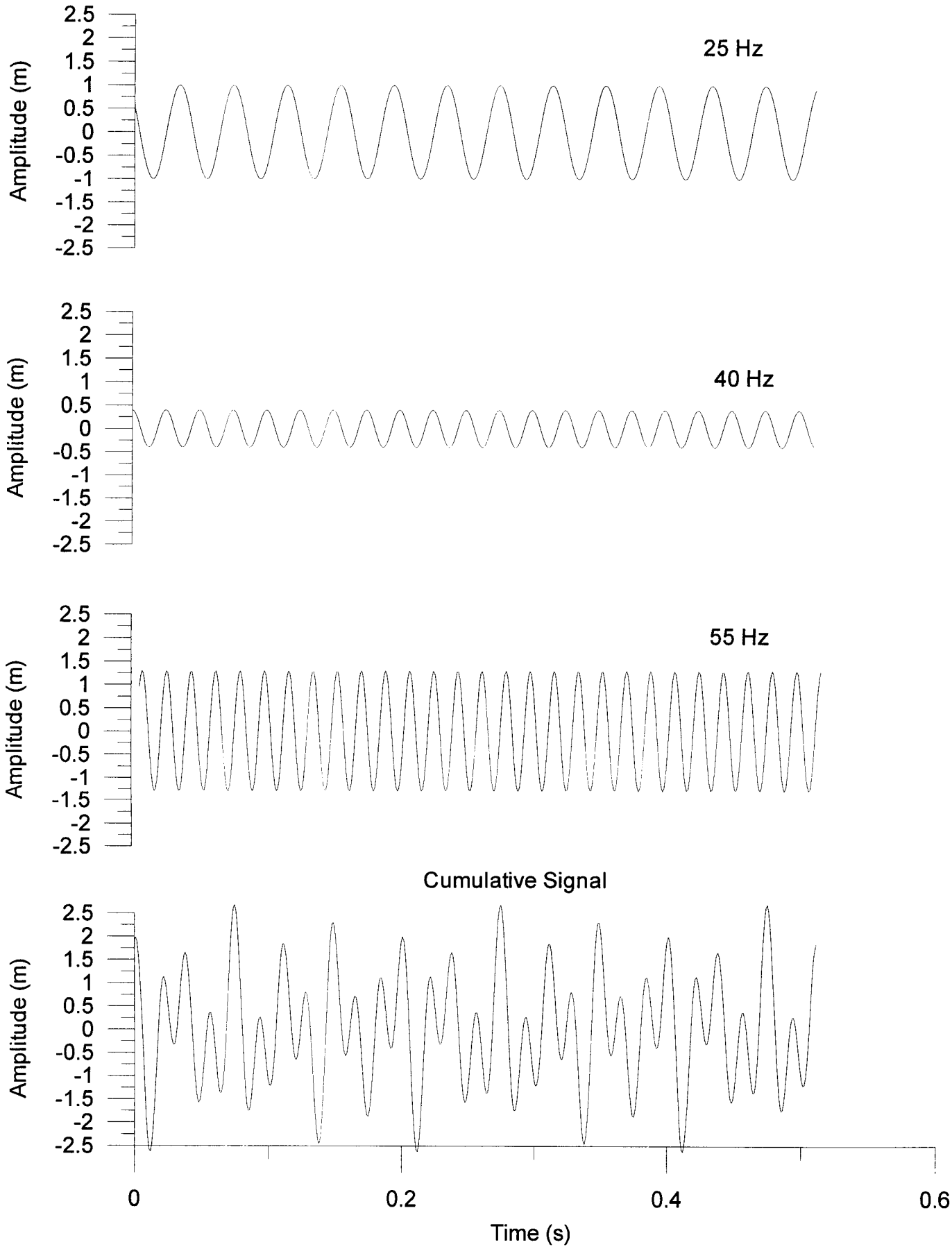
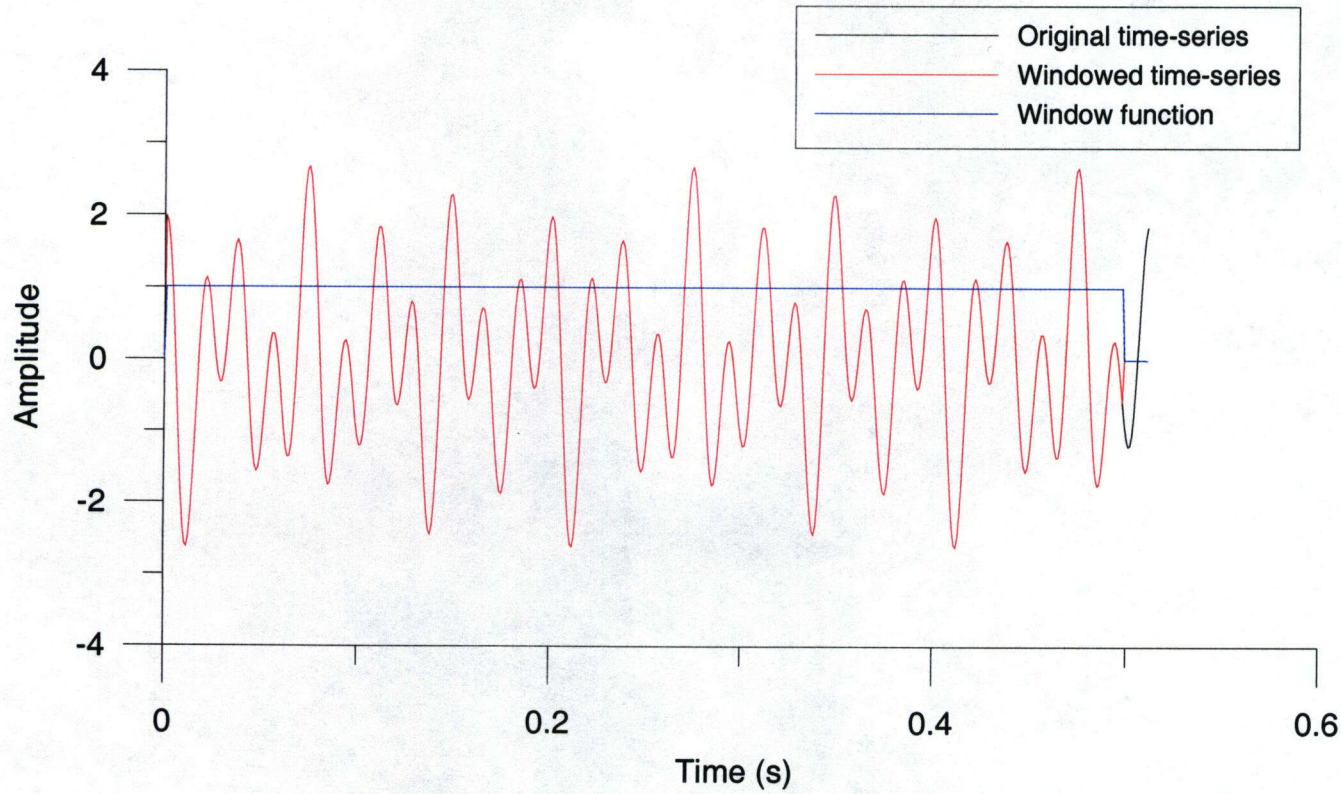

25 Hz
40 Hz
55 Hz
Cumulative Signal
Amplitude (m)
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
0.2
0.4
0.6
Time (s)

Fig 1

10/37

# Rectangular Window




# Spectrum

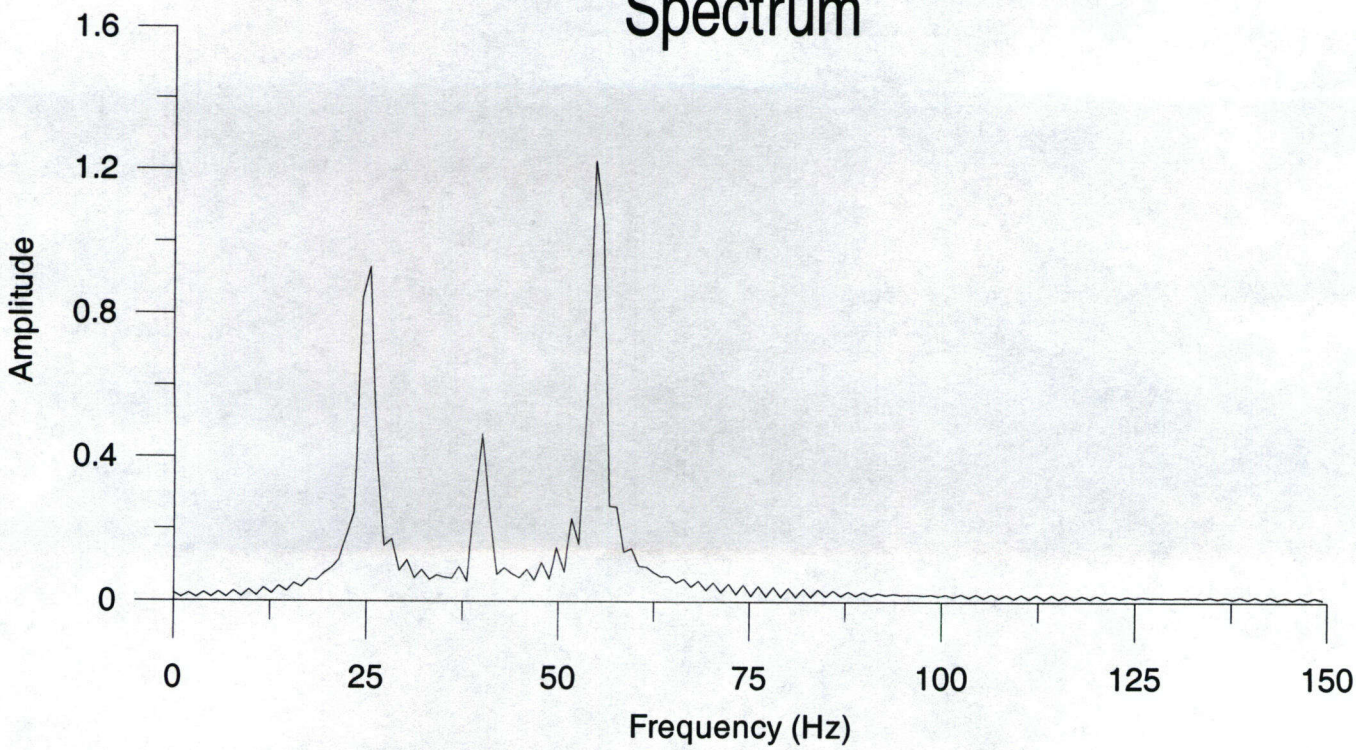
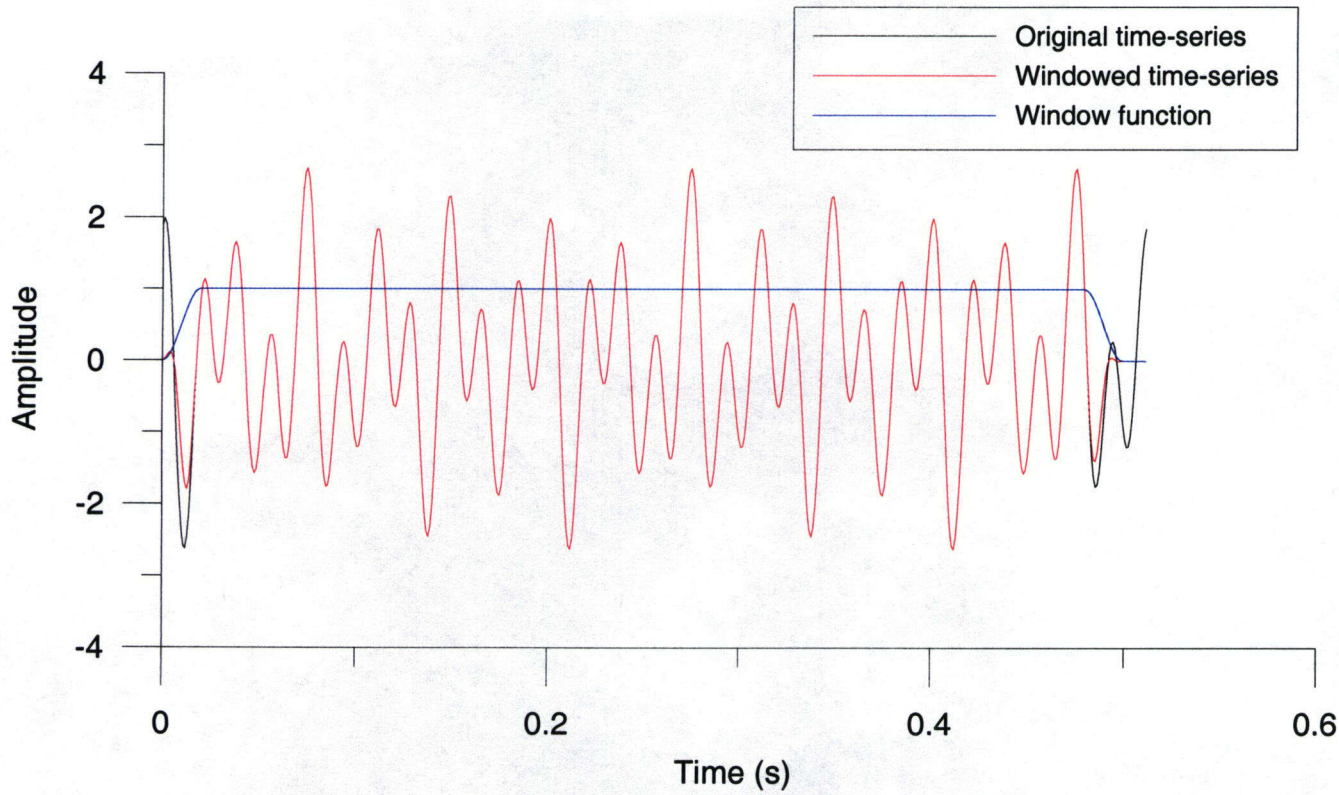



Fig 2

# Tapered Window



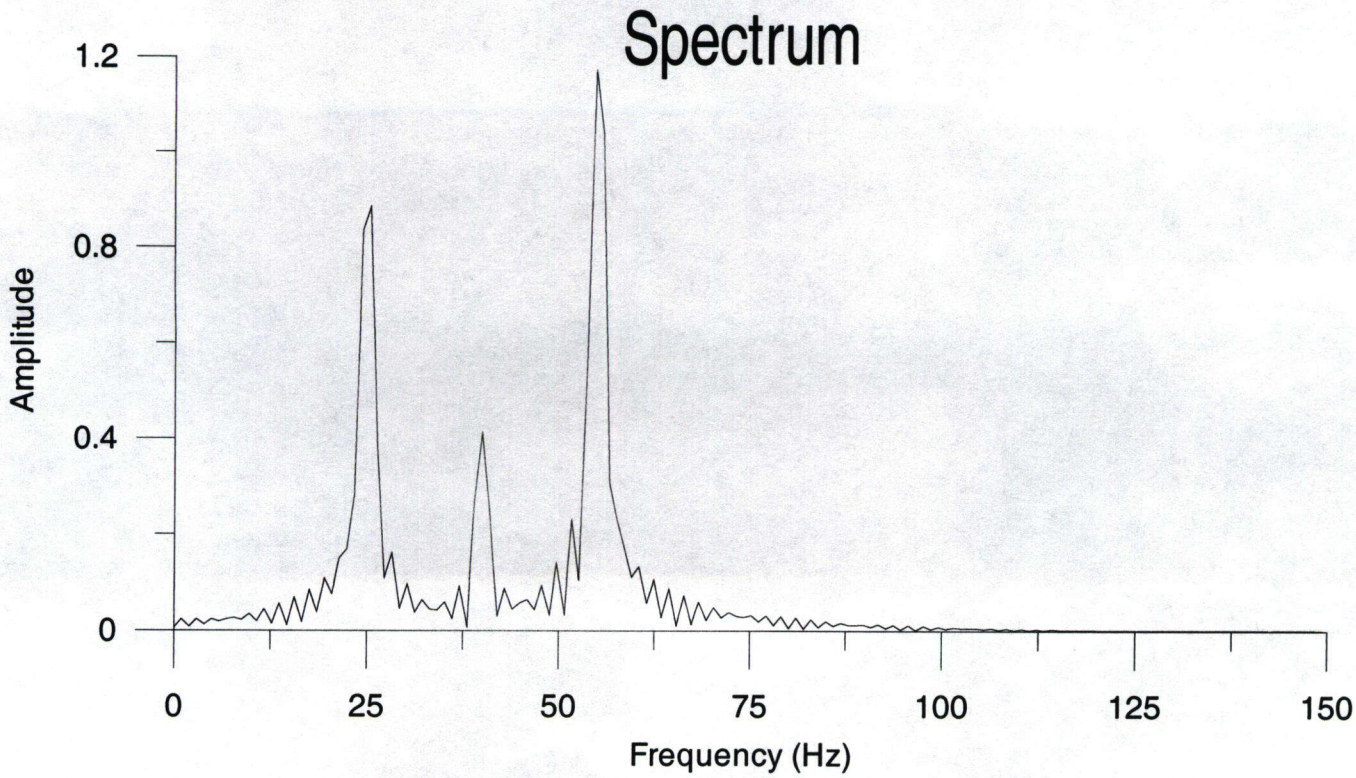
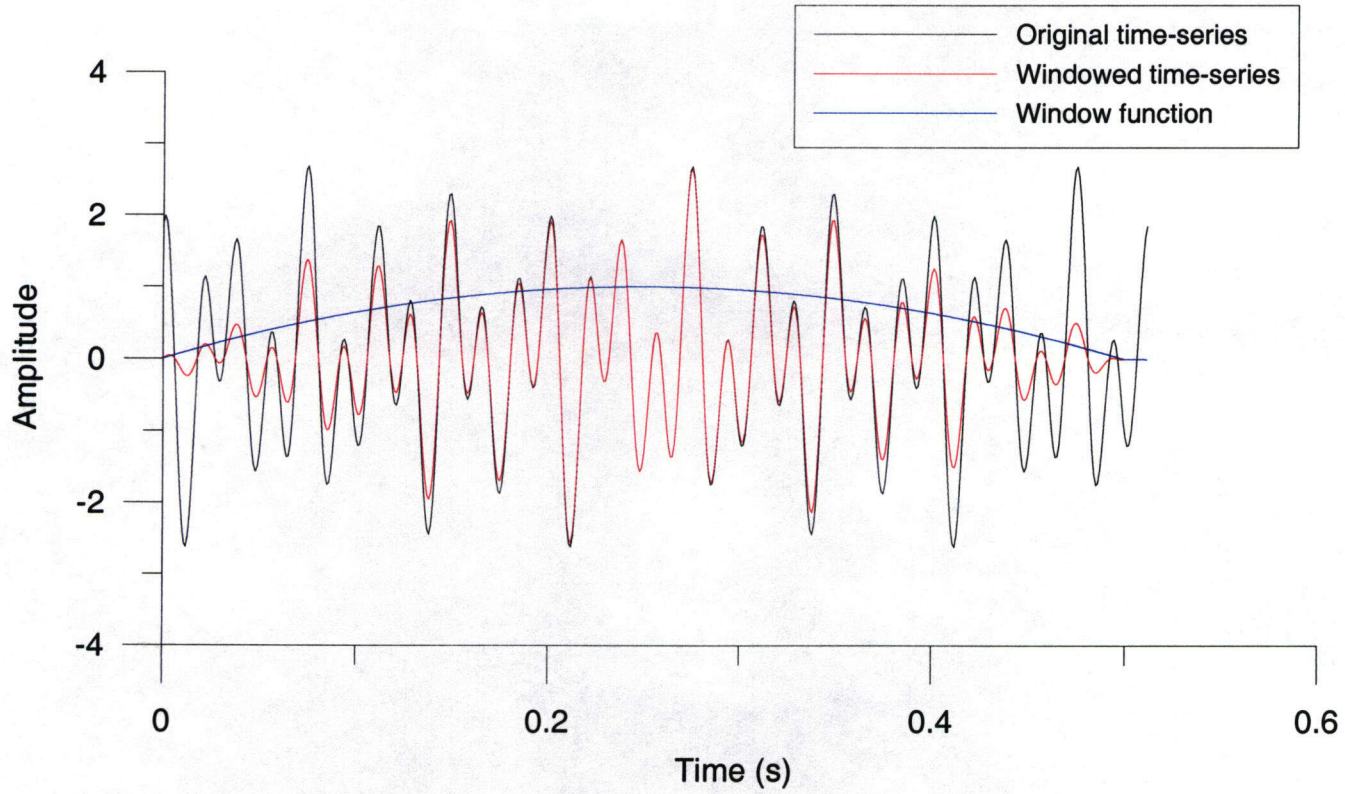




Fig 3

# Welch Window




# Spectrum

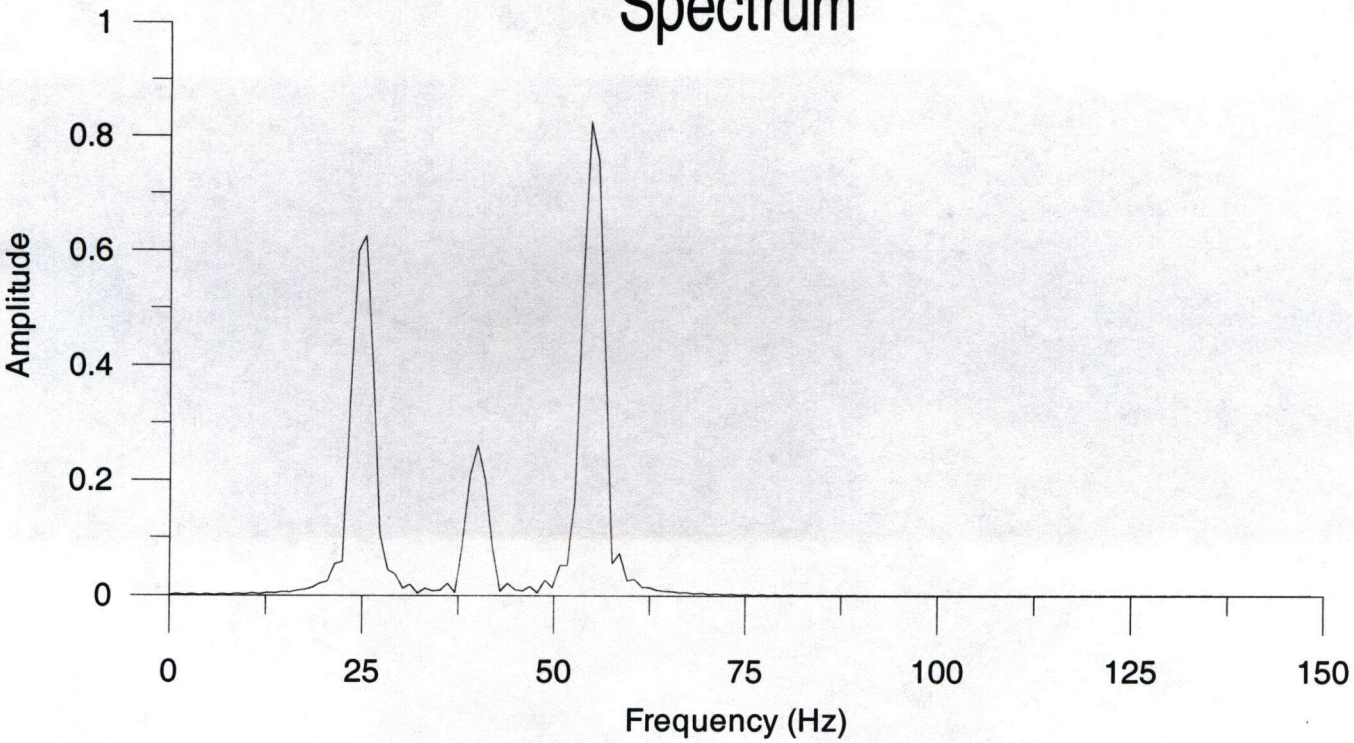
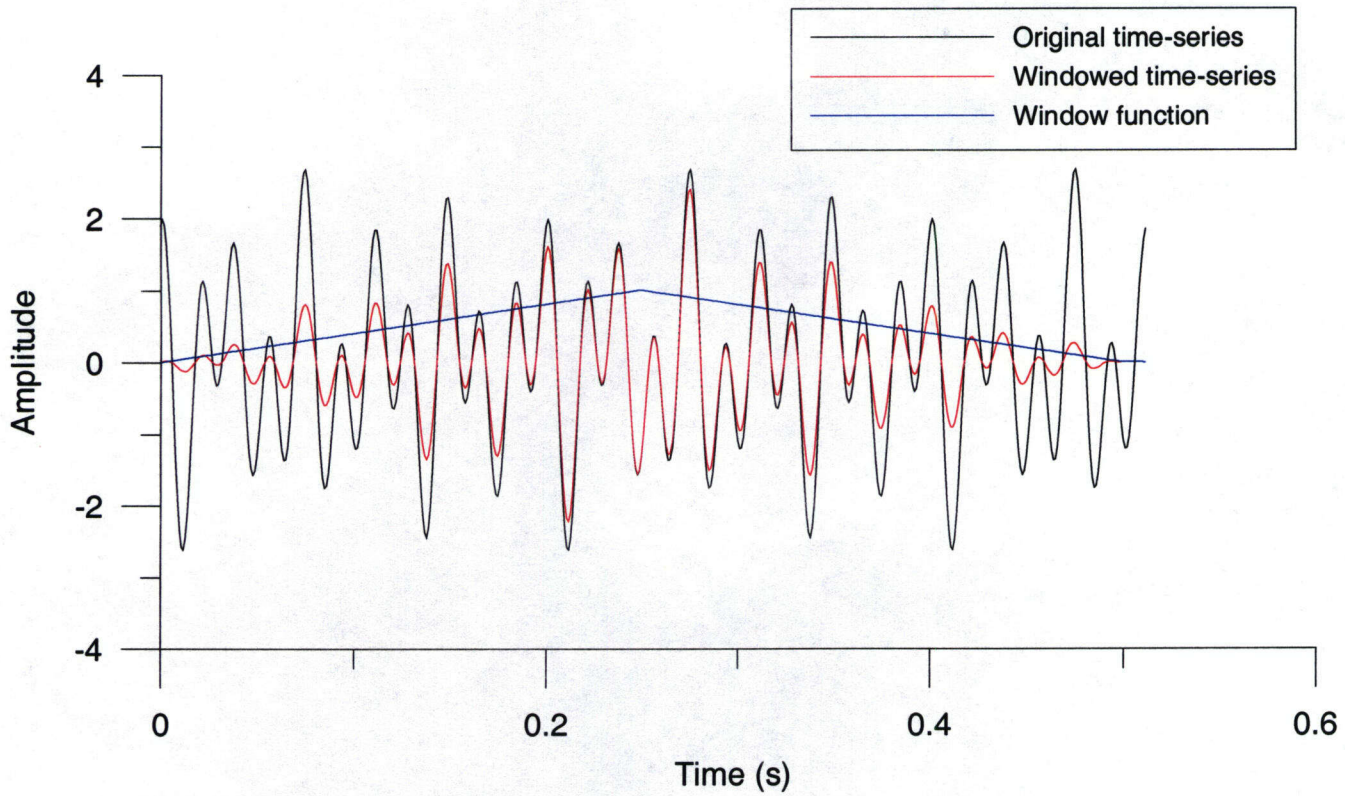



Fig 4

# Bartlett Window




## Spectrum

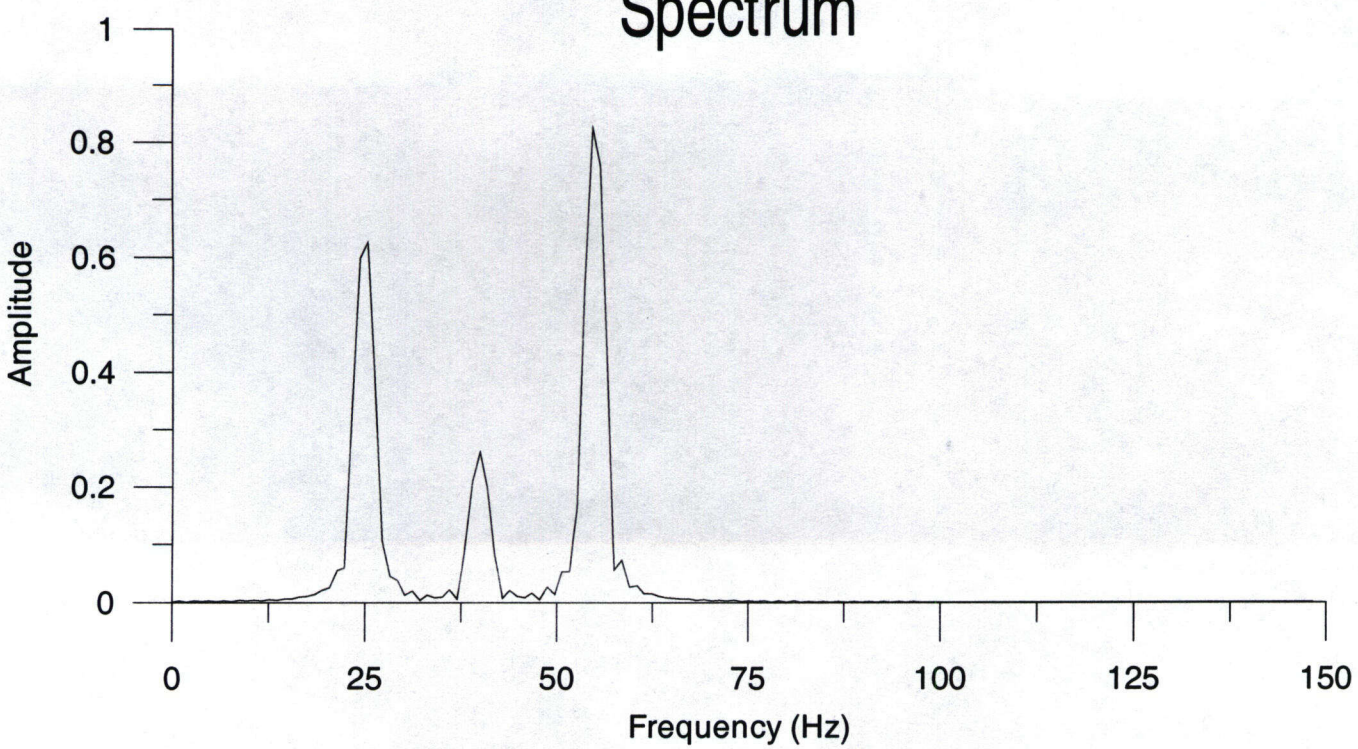
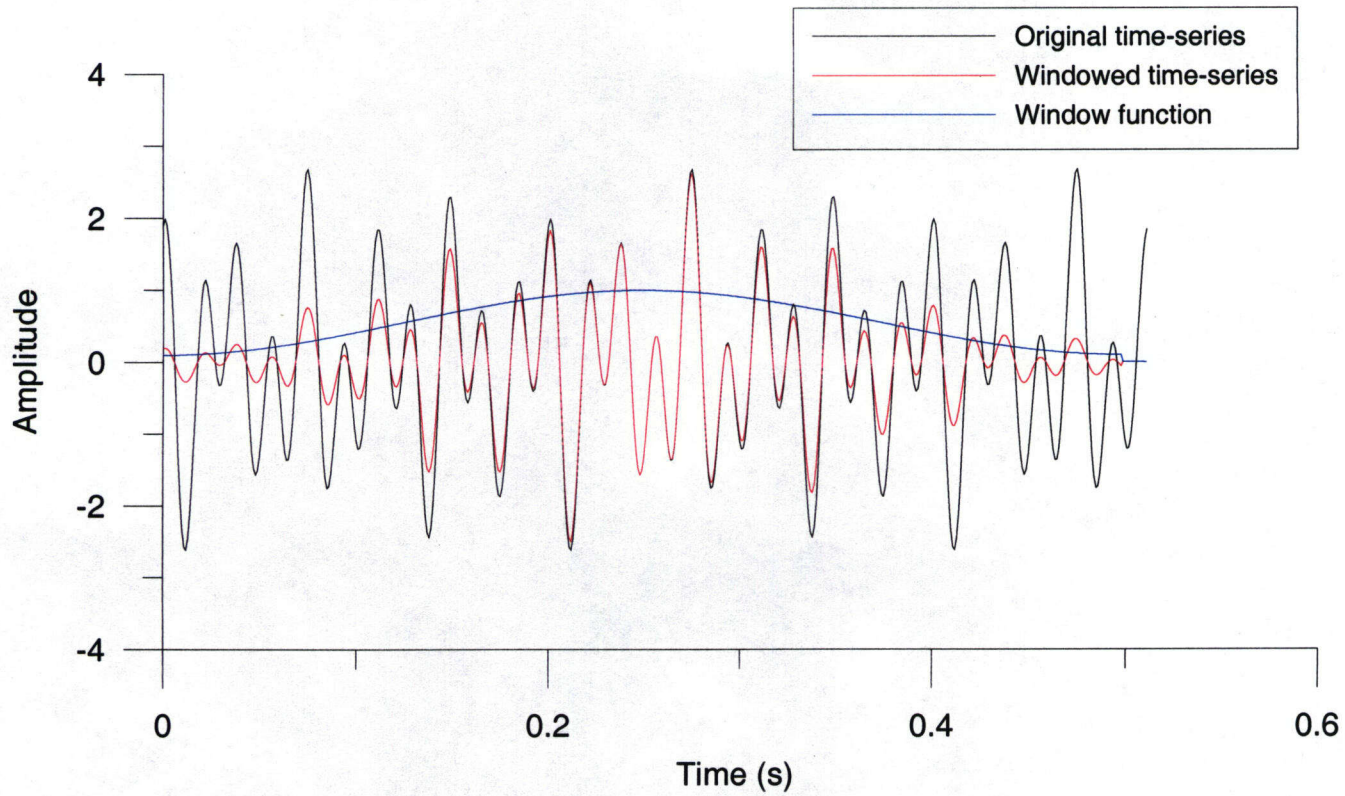



Fig 5

# Hamming Window




## Spectrum

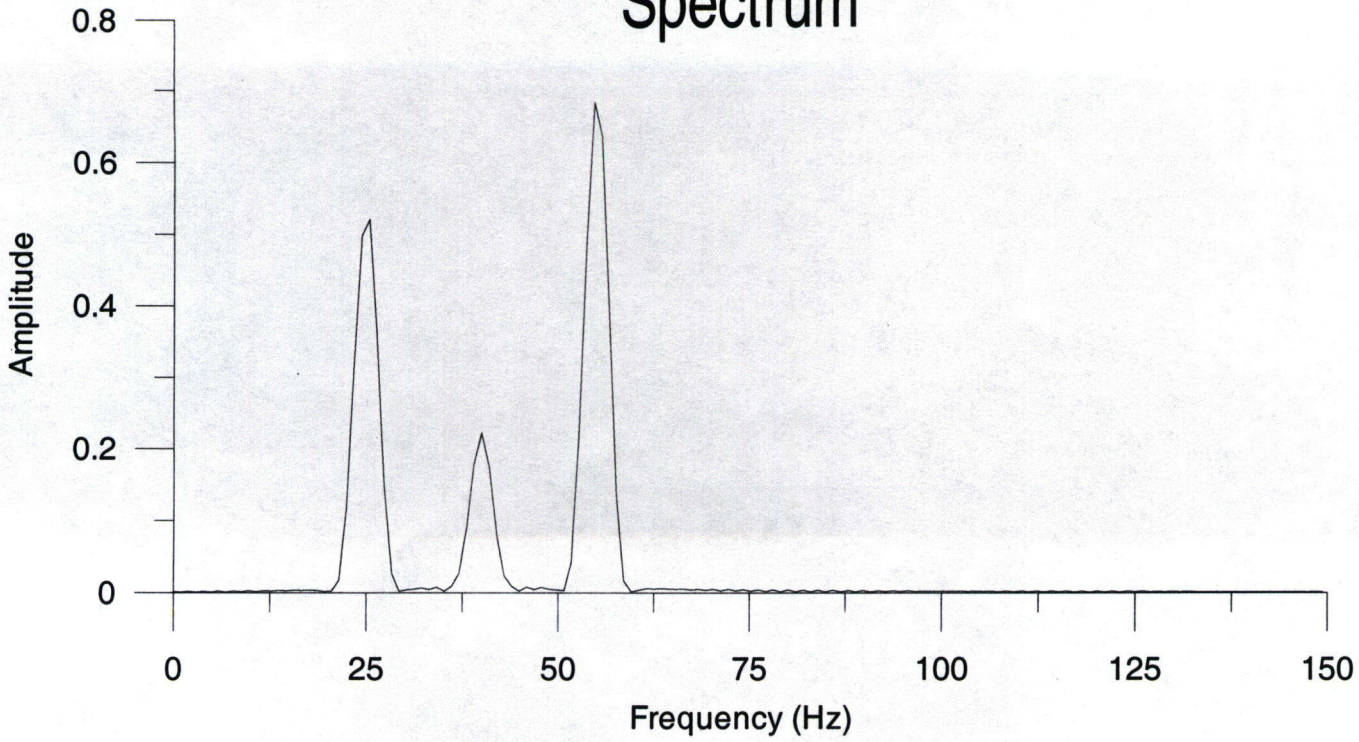
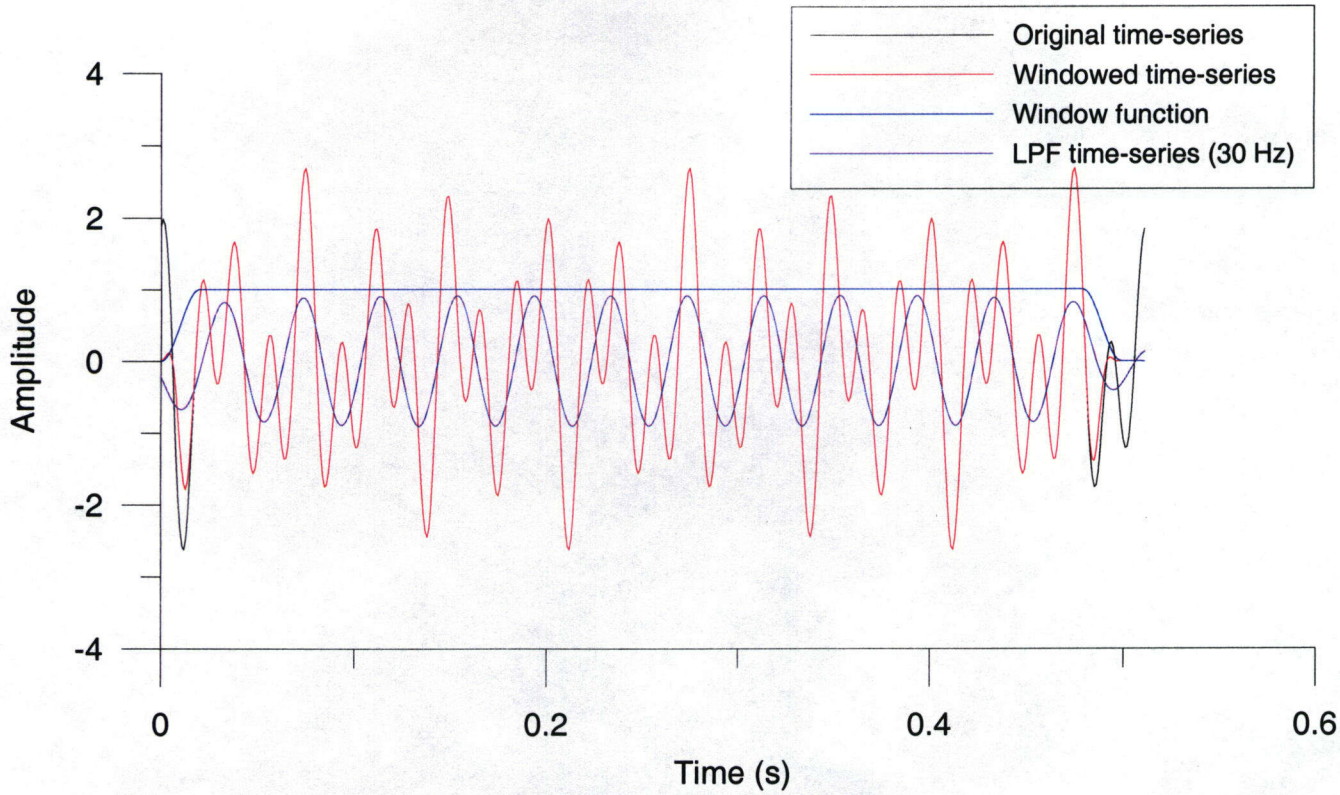



Fig 6

# Tapered Window Low-Pass Filter



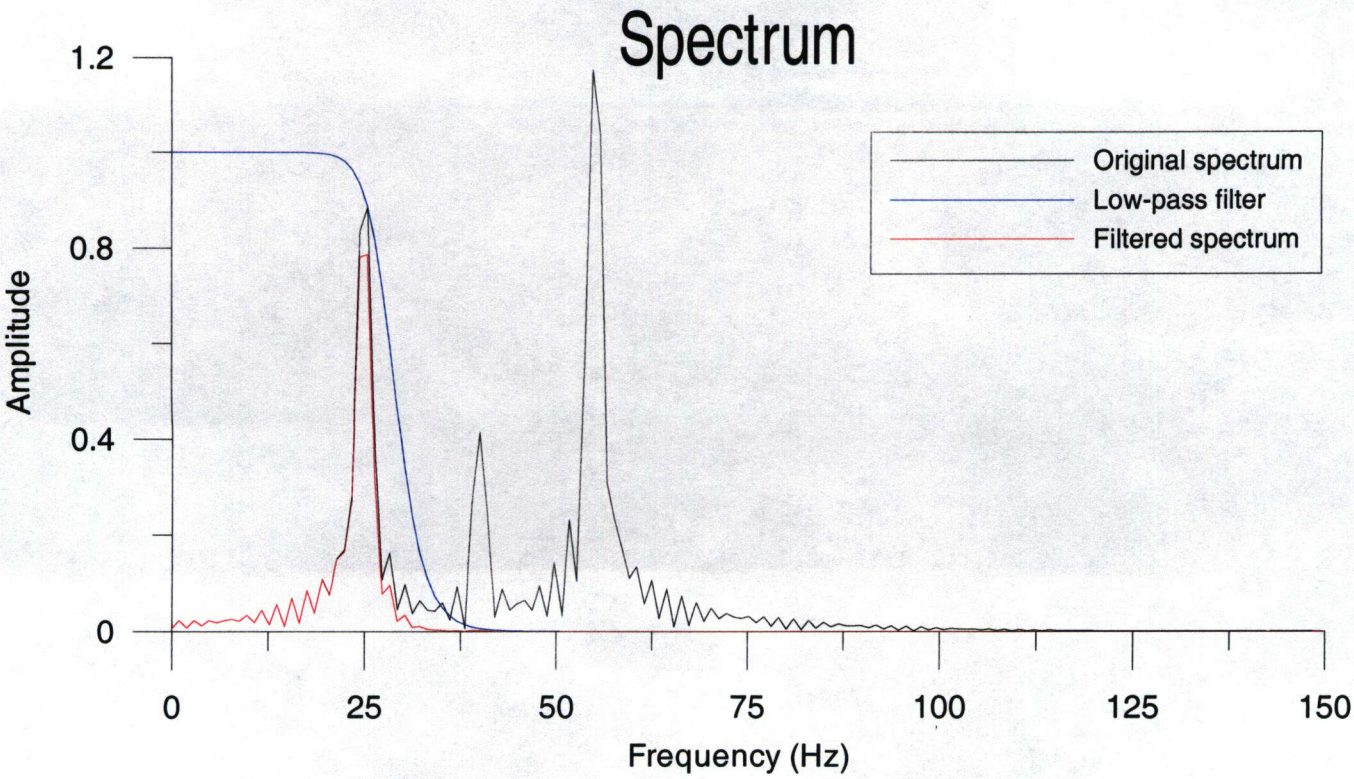
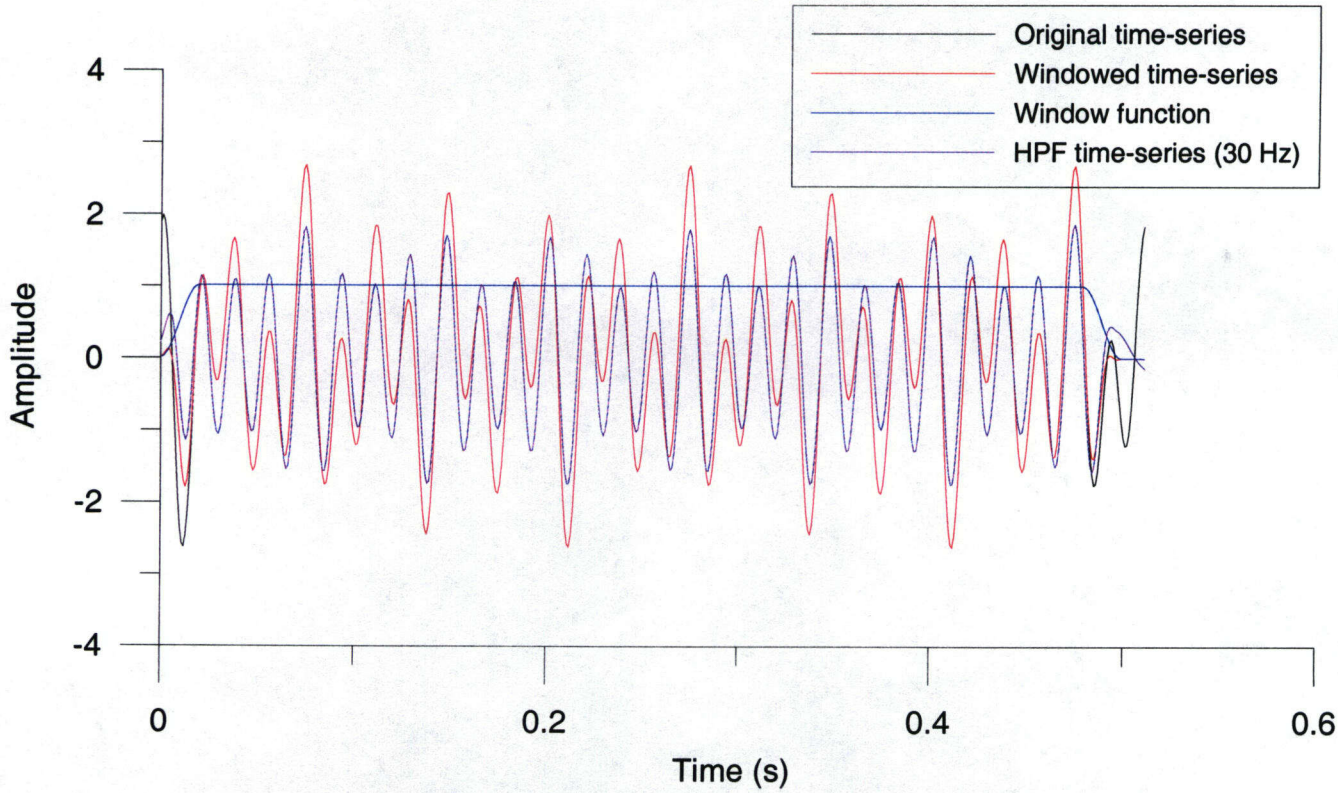




Fig7

# Tapered Window High-Pass Filter




## Spectrum

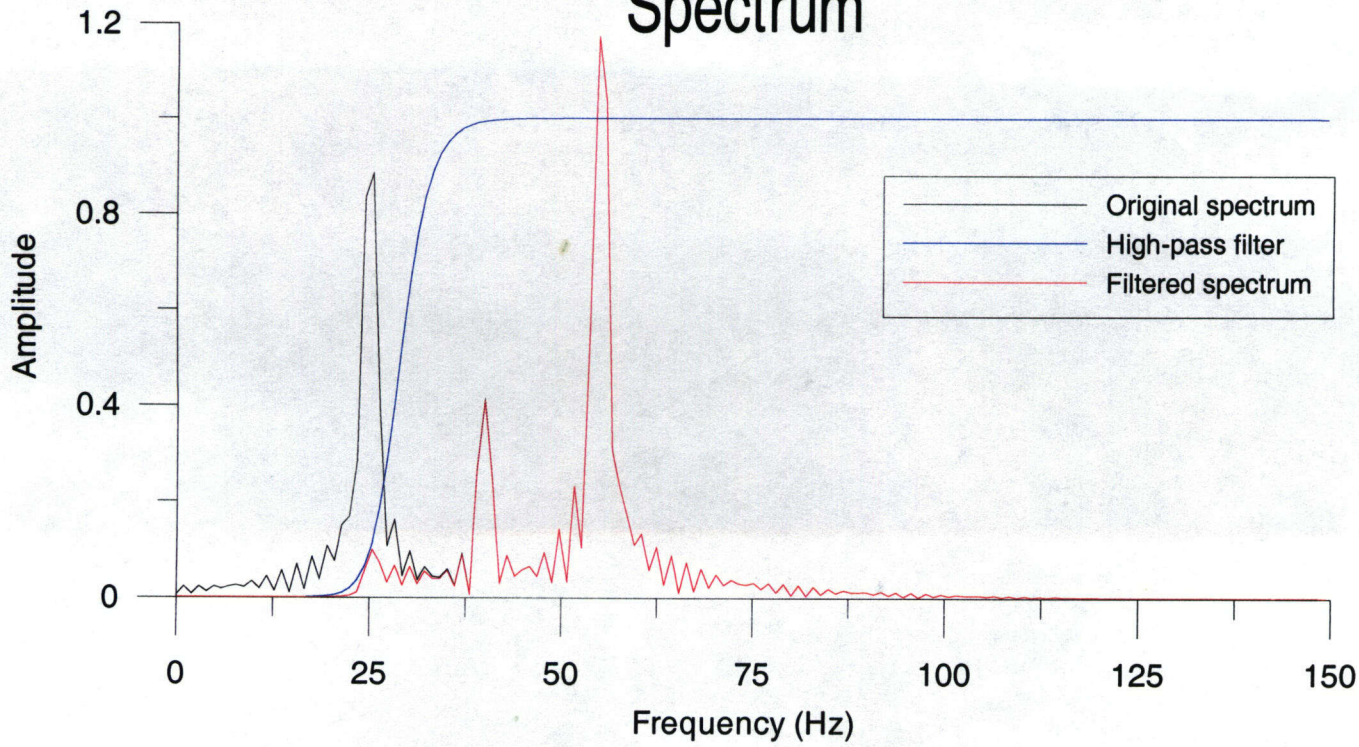
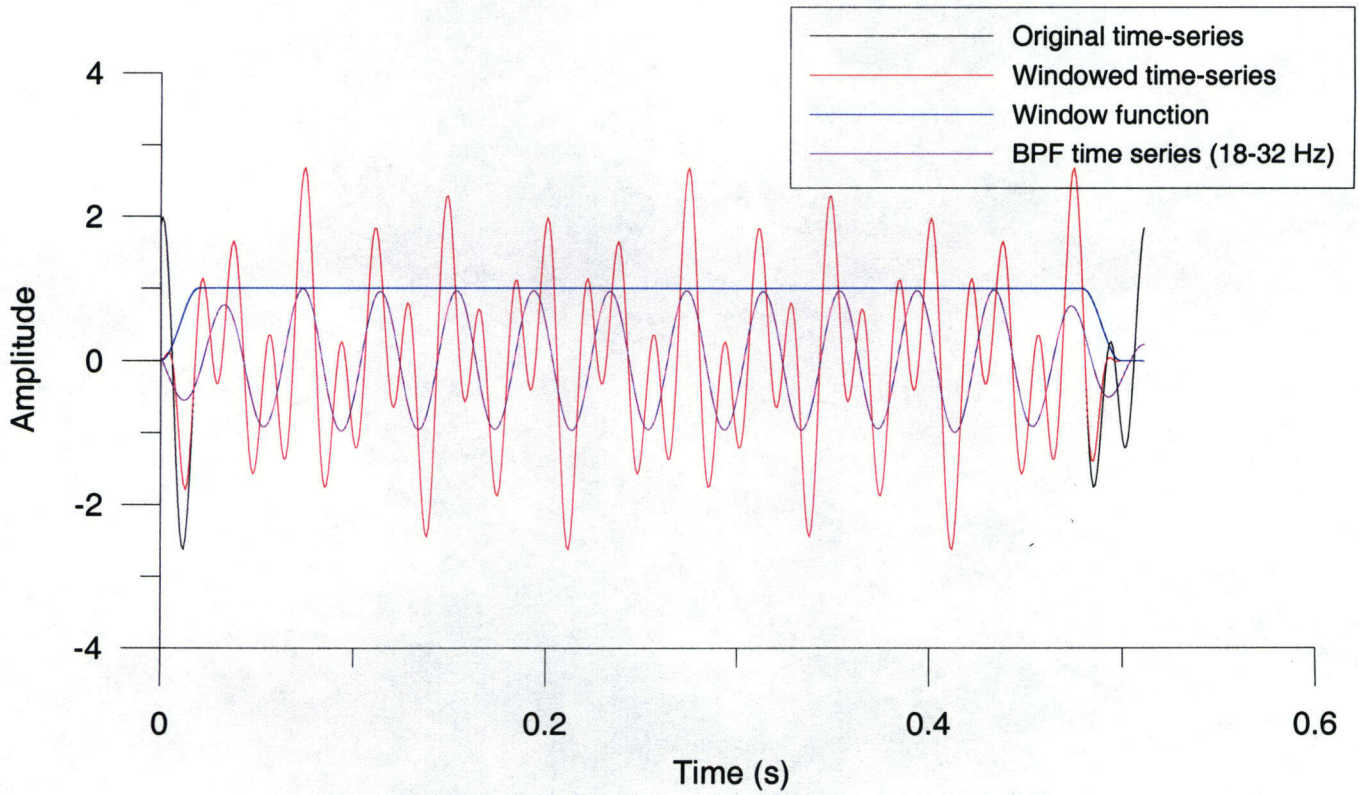



Fig 8

# Tapered Window Band-Pass Filter



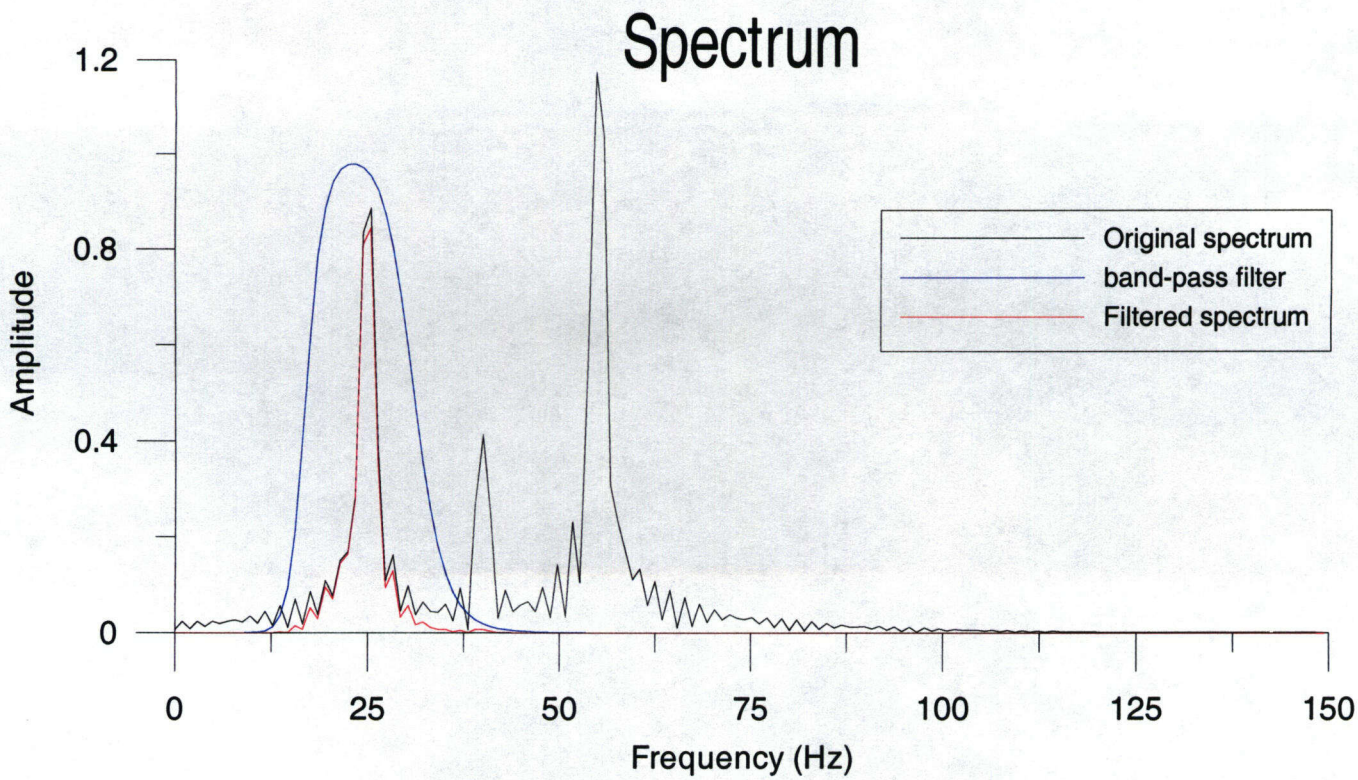
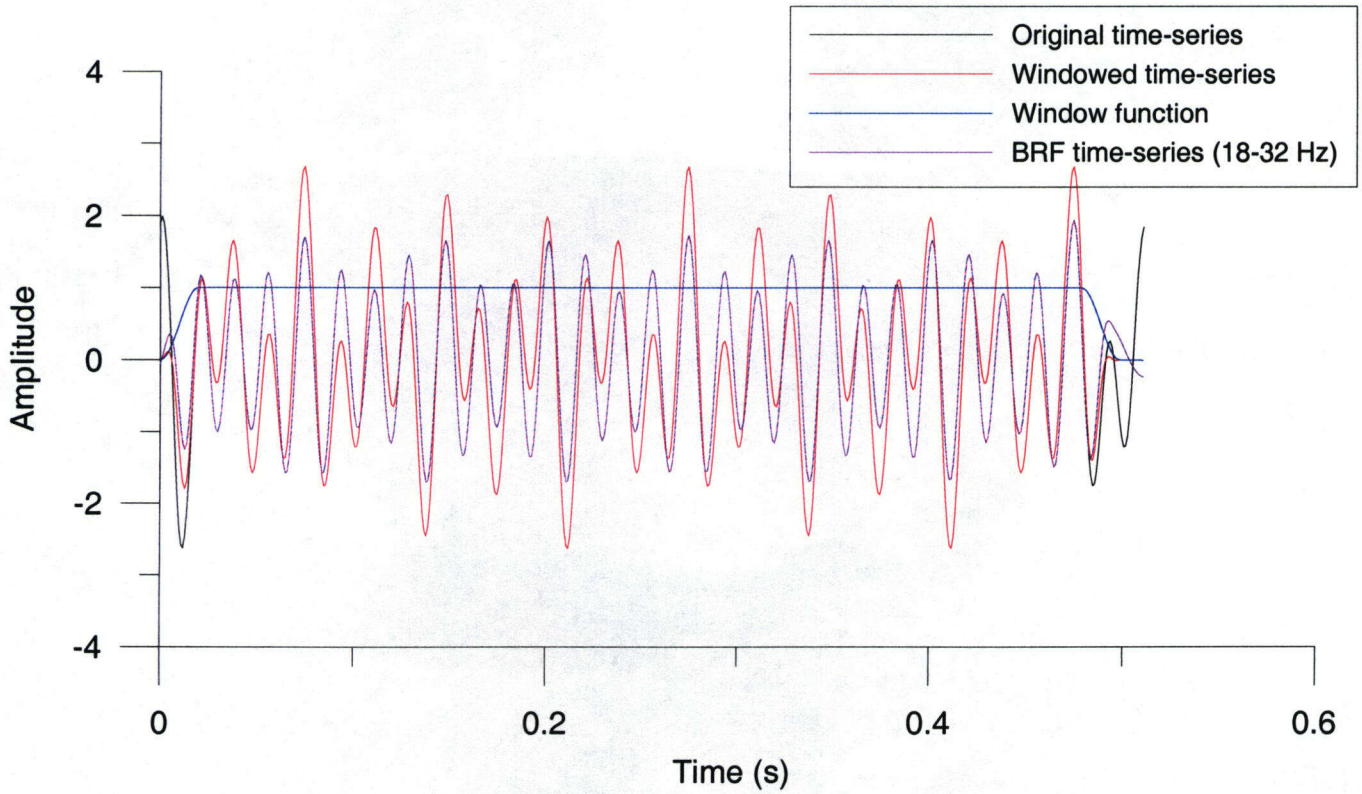




Fig 9

# Tapered Window Band-Remove Filter




## Spectrum

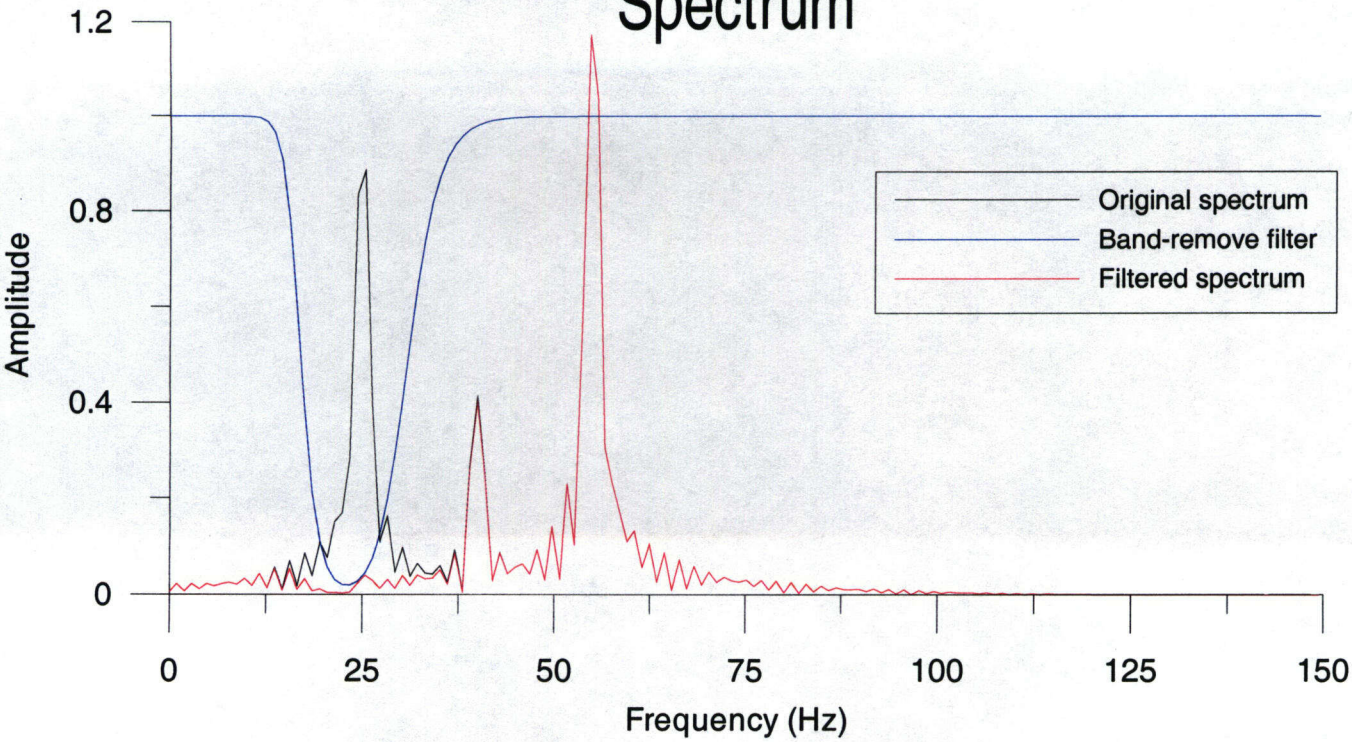
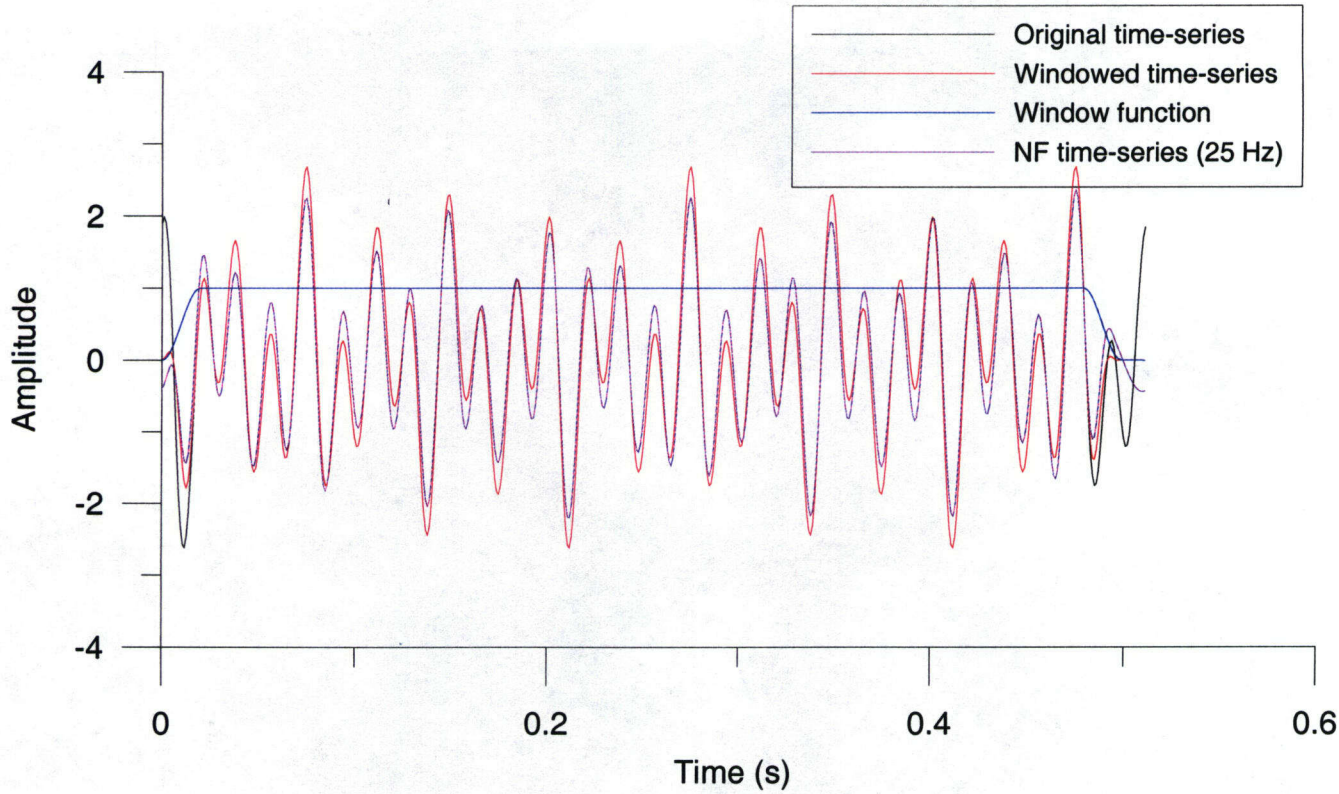



Fig 10

# Tapered Window Notch Filter




## Spectrum

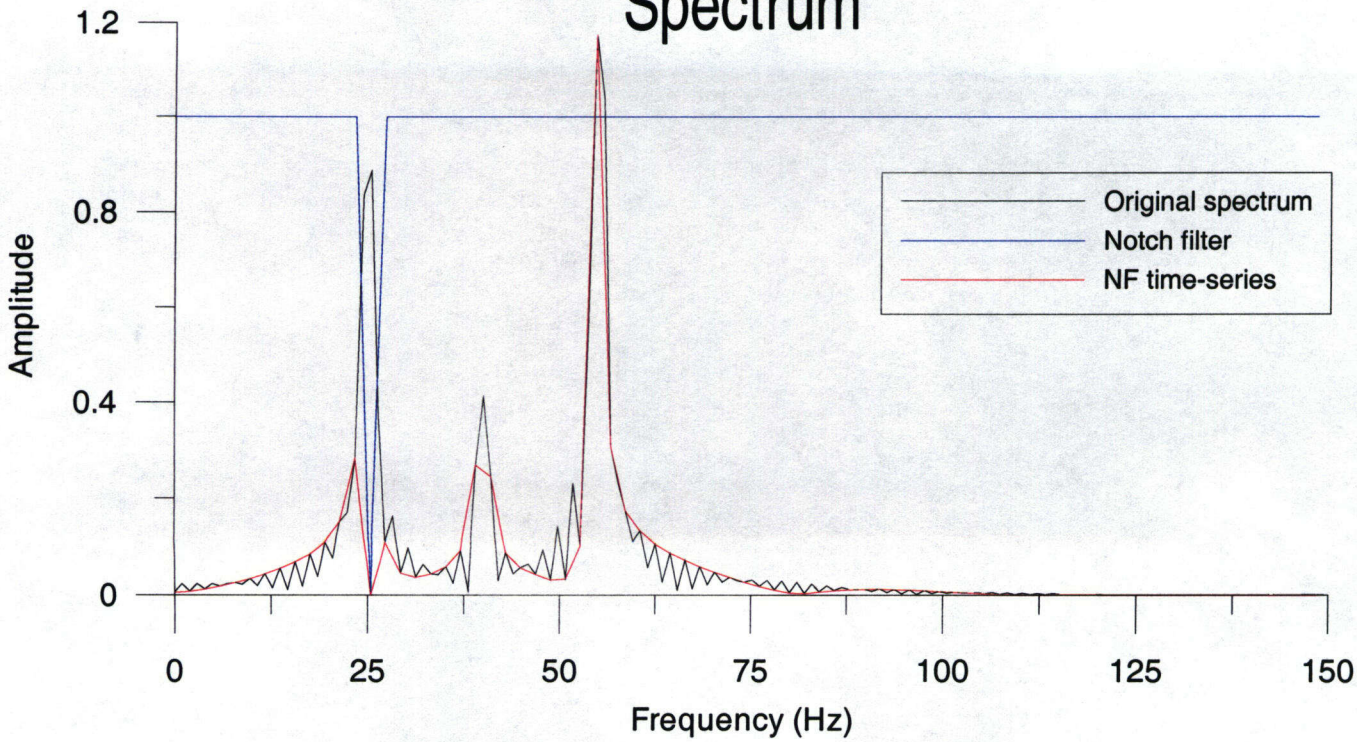



Fig 11

```
      program Fourier-filter_1D
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  This software is applicable for the analysis of time-series and 1D   c
c  space data such as gravity, magnetic and electromagnetic data.       c
c                                                                        c
c  Author: David Farrell                                                 c
c          CNWRA/SwRI                                                    c
c          6220 Culebra Rd                                               c
c          San Antonio, TX 78238                                         c
c          Tel: (210) 522-5208                                           c
c                                                                        c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      dimension timsers(80000),fil(80000),xseries(80000),xfrqser(80000)
      dimension freq(80000),ampt(80000),faze(80000),xfreser(80000)
      dimension xtim(80000),tm(80000),eamp(80000),rampt(80000),ts(80000)
      dimension ham_amp(80000), x_notch(80000), bart_amp(80000)
      dimension amp_notch(80000), tempf(80000),welch_amp(80000)
      dimension tap_amp(80000)

      complex xtimser,xfreser,xfrqser,xtim,xseries,x_notch
      character infil*20, freq_file*20, time_file*20
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      write(*,*) '*****************************************************'
      write(*,*) '*                  Fourier-filter 1D                *'
      write(*,*) '*                                                   *'
      write(*,*) '*  This software is applicable for the analysis of  *'
      write(*,*) '*  time-series and 1D space data such as gravity,   *'
      write(*,*) '*  magnetic and electromagnetic data.               *'
      write(*,*) '*                                                   *'
      write(*,*) '*  The software performs such tasks as:             *'
      write(*,*) '*  1. Data interpolation                            *'
      write(*,*) '*  2. Spectral analysis                             *'
      write(*,*) '*  3. Digital filtering                             *'
      write(*,*) '*  4. Convolution/deconvolution                     *'
      write(*,*) '*****************************************************'

      pause

 814  write(*,*) 'Pick the type of time series analysis to be performed'
      write(*,*) '0... See the spectrum for the data'
      write(*,*) '1... Bandpass filtering with Butterworth filter'
      write(*,*) '2... Notch filtering in the frequency domain'
      write(*,*) '3... Convolution'
      read(*,*) kopt
c
      if ((kopt.eq.1).or.(kopt.eq.0)) then
        write(*,*) '**************************'
        write(*,*) '*  Spectrum of the Data  *'
        write(*,*) '*   Bandpass Filtering   *'
        write(*,*) '**************************'
        write(*,*) ' '
        write(*,*) 'The input file is assumed to be composed of two'
        write(*,*) 'columns.'
        write(*,*) 'Column 1 ... time'
        write(*,*) 'Column 2 ... variable'
        write(*,*) 'enter the name of the input file'
        read(*,99) infil
 99     format(a20)

c............ Reading input time-series file ........................
        open(unit=1,file=infil,status='old')
        rewind 1
          iadd=0
 90       read(1,*,end=900) tt, ttt
          iadd=iadd+1
          tm(iadd)=tt
          timsers(iadd)=ttt
          go to 90
 900      continue
        close(unit=1)

c............ Interpolates the data if required ......................
        write(*,*) 'Is interpolation of the input data required?'
        write(*,*) '1 ... YES'
        write(*,*) '2 ... NO'
        read(*,*) intpl

        if (intpl.eq.1) then
          call interpolate(tm,timsers,iadd,iadd_c,dt)
          iadd=iadd_c
        else
          dt=tm(2)-tm(1)
        endif

c............ General parameters ....................................
        write(*,*) 'Enter conversion sign for FFT'
```

```
      write(*,*) '-1 ... Forward FFT'
      write(*,*) '+1 ... Inverse FFT'
      read(*,*) m_sign

      write(*,*) 'enter value for nlog'
      read(*,*) nlog
      lxs=2**nlog

      sr=1.0/dt
      delf=sr/lxs
      fnq=sr/2.0
      nfs=(lxs/2)+1

c............ Filter parameters ....................................
      write(*,*) '***********************'
      write(*,*) '*  Select the filter  *'
      write(*,*) '***********************'
      write(*,*) ' '
      write(*,*) '1.... High pass filter the data'
      write(*,*) '2.... Low pass filter the data'
      write(*,*) '3.... Pass a band of frequencies'
      write(*,*) '4.... Remove a band of frequencies'
      write(*,*) '5.... No filtering'
      read(*,*) i_filter

      if (i_filter.eq.2) then
        write(*,*) 'enter the low pass cutoff frequency'
        read(*,*) fcut_low
      endif

      if (i_filter.eq.1) then
        write(*,*) 'enter the high pass cutoff frequency'
        read(*,*) fcut_high
      endif

      if ((i_filter.eq.3).or.(i_filter.eq.4)) then
        write(*,*) 'enter the upper-cut off frequency'
        read(*,*) fcut_high
        write(*,*) 'enter the low-cut off frequency'
        read(*,*) fcut_low
          if (fcut_low.gt.fcut_high) then
            write(*,*) '*********************************************'
            write(*,*) '* Error ... lower cut-off frequency greater *'
            write(*,*) '* than upper cut-off frequency.             *'
            write(*,*) '*********************************************'
            stop
          endif
      endif
c
c............ Option for removing zero frequency term ...............
      avgvl=0.00D0
      write(*,*) 'do you want to remove average value from data'
      write(*,*) '1 ... YES'
      write(*,*) '2 ... NO'
      read(*,*) iopt

c............ Removing zero frequency term. Need to compute average ..
      if (iopt.eq.1) then
        sum=0.0D0

        do 961 ikk=1,iadd
961       sum=sum+timsers(ikk)

        avgvl=sum/float(iadd)

        do 962 ikk=1,iadd
962       timsers(ikk)=timsers(ikk)-avgvl
      endif
c
      write(*,*) 'average = ',avgvl

c............ Choosing the Windowing function ......................
      write(*,*) '**********************************'
      Write(*,*) '*  Select the windowing function  *'
      write(*,*) '**********************************'
      write(*,*) ' '
      write(*,*) '1.... Hamming Window'
      write(*,*) '2.... Welch Window'
      write(*,*) '3.... Bartlett Window'
      write(*,*) '4.... Tapered Rectangular Window'
      write(*,*) '5.... Rectangular Window'
      read(*,*) i_win


      if (i_win.eq.1) then
c............ Applying the Hamming Window ........................
        write(*,*) 'apply Hamming Window'
        write(*,*) 'enter the start time for the Hamming Window'
```

```
          read(*,*) start_win
          write(*,*) 'enter the window width (time) of the Hamming'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Hamming Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Hamming.out  *'
          write(*,*) '************************************'

          open(unit=10,file="hamming.out",status='unknown')
          rewind 10
             call hamming(istart_win,iwin_width,ham_amp,iadd)
             do ipk=1,iadd
               timsers(ipk)=timsers(ipk)*ham_amp(ipk)
               if (ipk.le.iadd) then
                 write(10,*) tm(ipk), timsers(ipk), ham_amp(ipk)
               endif
             enddo
          close(unit=10)
        endif
c

        if (i_win.eq.2) then
c............ Applying the Welch Window ............................
          write(*,*) 'apply Welch Window'
          write(*,*) 'enter the start time for the Welch Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Welch'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Welch Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '**********************************'
          write(*,*) '*  WINDOWING FILE --> Welch.out  *'
          write(*,*) '**********************************'

          open(unit=10,file="Welch.out",status='unknown')
          rewind 10
             call welch_window(istart_win,iwin_width,welch_amp,iadd)
             do ipk=1,iadd
               timsers(ipk)=timsers(ipk)*welch_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), welch_amp(ipk)
              endif
             enddo
          close(unit=10)
        endif
c

        if (i_win.eq.3) then
c............ Applying the Bartlett Window ........................
          write(*,*) 'apply Bartlett Window'
          write(*,*) 'enter the start time for the Bartlett Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Bartlett'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Bartlett Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Bartlett.out  *'
          write(*,*) '************************************'

          open(unit=10,file="Bartlett.out",status='unknown')
          rewind 10
            call bartlett_window(istart_win,iwin_width,bart_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*bart_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), bart_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif
c

        if (i_win.eq.4) then
c............ Applying the Tapered Rectangular Window ...............
          write(*,*) 'apply Tapered Window'
          write(*,*) 'enter the start time for the Tapered Window'
          read(*,*) start_win
```

```
          write(*,*) 'enter the window width (time) for the Tapered'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Tapered Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Tapered.out  *'
          write(*,*) '************************************'

          open(unit=10,file="Tapered.out",status='unknown')
          rewind 10
            call tapered_window(istart_win,iwin_width,tap_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*tap_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), tap_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif


        if (i_win.eq.5) then
c............ Applying the Rectangular Window ......................
          write(*,*) 'apply Tapered Window'
          write(*,*) 'enter the start time for the Rectangular Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Rectangular'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Tapered Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Rectang.out  *'
          write(*,*) '************************************'

          open(unit=10,file="Rectang.out",status='unknown')
          rewind 10
            call rectang_window(istart_win,iwin_width,tap_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*tap_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), tap_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif


c............ Padding the end of the data set ......................
        do 97 ikk=iadd+1,lxs
          timsers(ikk)=0.00D0
 97       tm(ikk)=tm(ikk-1)+dt

c............ Forward transform ....................................
        call freqf(nfs,delf,freq)
        call datax(iadd,timsers,lxs,xseries)
        call nlogn(nlog,xseries,m_sign)
        call xspect(lxs,xseries,nfs,ampt,faze)

c............ Normalizing the amplitude to account for ..............
c............ transform scaling artifacts ...........................
        do 756 k=1,nfs
 756      ampt(k)=ampt(k)/float(iadd/2)

c............ Computing the energy spectrum .........................
        do 757 k=1,nfs
 757      eamp(k)=ampt(k)**2

c............ Normalizing the energy spectrum ......................
c        eampmx=eamp(1)
c        do 758 k=2,nfs
c          if(eamp(k).gt.eampmx) eampmx=eamp(k)
c 758      continue

c        do 776 k=1,nfs
c          eamp(k)=eamp(k)/eampmx
c 776    continue

c............ Normalizing the amplitude spectrum ....................
         ampmax=1.0D0
c         ampmax=ampt(1)
c         do 760 k=2,nfs
c           if (ampt(k).gt.ampmax) ampmax=ampt(k)
```

```
c 760     continue
c
         do 761 k=1,nfs
           rampt(k)=ampt(k)/ampmax
 761     continue

c............ Writing to output file if only data spectrum required ..
        if (kopt.eq.0) then
          write(*,*) 'Spectrum written to file SPECTRUM.DAT'
          open(unit=23,file='spectrum.dat',status='unknown')
          rewind 23
            do 159 k=1,nfs
 159          write(23,*) freq(k), rampt(k), faze(k)
          close(unit=23)
        endif

c............ Low-pass filtering ...................................
        if(i_filter.eq.2) then
          nmb=1
          norder=8
          call bfilter(fcut_low,nfs,nmb,fil,norder,delf)
          do 762 k=1,nfs
 762        rampt(k)=rampt(k)*fil(k)

          open(unit=15,file='low_pass_filter.dat',status='unknown')
          rewind 15
            do 307 i=1,nfs
 307          write(15,*) freq(i), fil(i), rampt(i)
          close(unit=15)
        endif

c............ High-pass filtering ..................................
        if(i_filter.eq.1) then
          nmb=1
          norder=8
          call bfilter(fcut_high,nfs,nmb,fil,norder,delf)
          do 765 k=1,nfs
 765        fil(k)=1.0-fil(k)
          do 764 k=1,nfs
 764        rampt(k)=rampt(k)*fil(k)

          open(unit=16,file='high_pass_filter.dat',status='unknown')
          rewind 16
            do 308 i=1,nfs
 308          write(16,*) freq(i), fil(i), rampt(i)
          close(unit=16)
        endif

c............ Band-pass filtering ...................................
        if(i_filter.eq.3) then
          nmb=1
          norder=8
c   Low-pass filtering
          call bfilter(fcut_high,nfs,nmb,fil,norder,delf)
          do 913 k=1,nfs
            tempf(k)=fil(k)
 913        fil(k)=0.0
c   High-pass filtering
          call bfilter(fcut_low,nfs,nmb,fil,norder,delf)
          do 914 k=1,nfs
 914        fil(k)=(1.0-fil(k))*tempf(k)
          do 916 k=1,nfs
 916        rampt(k)=rampt(k)*fil(k)

          open(unit=17,file='band_pass_filter.dat',status='unknown')
          rewind 17
            do 309 i=1,nfs
 309          write(17,*) freq(i), fil(i), rampt(i)
          close(unit=17)
        endif

c............ Band-remove filtering ..................................
        if(i_filter.eq.4) then
          nmb=1
          norder=8
c   Low-pass filtering
          call bfilter(fcut_high,nfs,nmb,fil,norder,delf)
          do 243 k=1,nfs
            tempf(k)=fil(k)
 243        fil(k)=0.0
c   High-pass filtering
          call bfilter(fcut_low,nfs,nmb,fil,norder,delf)
          do 244 k=1,nfs
 244        fil(k)=1.0-fil(k)
          do 287 k=1,nfs
 287        fil(k)=abs(1.0-(fil(k)*tempf(k)))
          do 246 k=1,nfs
 246        rampt(k)=rampt(k)*fil(k)
```

```
          open(unit=18,file='band_remov filter.dat',status='unknown')
          rewind 18
            do 318 i=1,nfs
 318          write(18,*) freq(i), fil(i), rampt(i)
          close(unit=18)
        endif

c............ Performing the inverse Fourier Transform ..............
        m_sign=-m_sign
        do 763 k=1,nfs
 763      rampt(k)=rampt(k)*(FLOAT(IADD)/2.0)

        call spectx(nfs,rampt,faze,lxs,xseries)
        call nlogn(nlog,xseries,m_sign)
        call xdata(lxs,xseries,ts)
c

      else if (kopt.eq.2) then

c============= Performing Notch filtering ============================

        write(*,*) '*********************'
        write(*,*) '*  Notch Filtering  *'
        write(*,*) '*********************'
        write(*,*) ' '
        write(*,*) '##################################################'
        write(*,*) '#  Before executing this option it is advisable  #'
        write(*,*) '#  to first look at the spectrum of the data to  #'
        write(*,*) '#  the frequency to be notched                   #'
        write(*,*) '##################################################'

        pause

 127    write(*,*) 'Do you wish to continue with this option?'
        write(*,*) '1 ... YES'
        write(*,*) '2 ... NO'
        read(*,*) notch_opt

        if (notch_opt.eq.1) then
          go to 128
        else if (notch_opt.eq.2) then
          go to 814
        else
          write(*,*) 'Invalid option'
          goto 127
        endif

 128    write(*,*) 'enter the name of the input file'
        read(*,99) infil

c............ Reading input time-series file ........................
        open(unit=1,file=infil,status='old')
        rewind 1
          iadd=0
 1090     read(1,*,end=990) tt, ttt
          iadd=iadd+1
          tm(iadd)=tt
          timsers(iadd)=ttt
          go to 1090
 990      continue
        close(unit=1)

c............ Interpolates the data if required ......................
        write(*,*) 'Is interpolation of the input data required?'
        write(*,*) '1 ... YES'
        write(*,*) '2 ... NO'
        read(*,*) intpl

        if (intpl.eq.1) then
          call interpolate(tm,timsers,iadd,iadd_c,dt)
          iadd=iadd_c
        else
          dt=tm(2)-tm(1)
        endif

c............ General parameters ....................................
        write(*,*) 'Enter conversion sign for FFT'
        write(*,*) '-1 ... Forward FFT'
        write(*,*) '+1 ... Inverse FFT'
        read(*,*) m_sign

        write(*,*) 'enter value for nlog'
        read(*,*) nlog
        lxs=2**nlog

        write(*,*) 'Enter control parameter for sharpness of the notch'
        write(*,*) 'Default = 0.00001  but you should experiment'
```

```
      read(*,*) eta

      sr=1.0/dt
      delf=sr/lxs
      fnq=sr/2.0
      nfs=(lxs/2)+1

c............ Get the frequency to be removed ......................
      write(*,*) 'Enter the frequency to be removed'
      read(*,*) f0

c............ Option for removing zero frequency term ...............
      avgvl=0.00D0
      write(*,*) 'do you want to remove average value from data'
      write(*,*) '1 ... YES'
      write(*,*) '2 ... NO'
      read(*,*) iopt

c............ Removing zero frequency term. Need to compute average ..
      if (iopt.eq.1) then
        sum=0.0D0

        do 971 ikk=1,iadd
 971      sum=sum+timsers(ikk)

          avgvl=sum/float(iadd)

        do 972 ikk=1,iadd
 972      timsers(ikk)=timsers(ikk)-avgvl
      endif
c
      write(*,*) 'average = ',avgvl
c
c............ Choosing the Windowing function ......................
      write(*,*) '*********************************'
      Write(*,*) '*  Select the windowing function  *'
      write(*,*) '*********************************'
      write(*,*) ' '
      write(*,*) '1.... Hamming Window'
      write(*,*) '2.... Welch Window'
      write(*,*) '3.... Bartlett window'
      write(*,*) '4.... Tapered Rectangular Window'
      write(*,*) '5.... Rectangular Window'
      read(*,*) i_win

      if (i_win.eq.1) then
c............ Applying the Hamming Window ........................
        write(*,*) 'apply Hamming Window'
        write(*,*) 'enter the start time for the Hamming Window'
        read(*,*) start_win
        write(*,*) 'enter the window width (time) of the Hamming'
        write(*,*) 'Window'
        read(*,*) win_width
c Computing the starting and ending points of the Hamming Window
        istart_win=int(start_win/dt)
        iwin_width=int(win_width/dt)

        write(*,*) '*********************************'
        write(*,*) '*  WINDOWING FILE --> Hamming.out  *'
        write(*,*) '*********************************'

        open(unit=10,file="hamming.out",status='unknown')
        rewind 10
           call hamming(istart_win,iwin_width,ham_amp,iadd)
           do ipk=1,iadd
             timsers(ipk)=timsers(ipk)*ham_amp(ipk)
             if (ipk.le.iadd) then
               write(10,*) tm(ipk), timsers(ipk), ham_amp(ipk)
             endif
           enddo
        close(unit=10)
      endif


      if (i_win.eq.2) then
c............ Applying the Welch Window ............................
        write(*,*) 'apply Welch Window'
        write(*,*) 'enter the start time for the Welch Window'
        read(*,*) start_win
        write(*,*) 'enter the window width (time) for the Welch'
        write(*,*) 'Window'
        read(*,*) win_width
c Computing the starting and ending points of the Welch Window
        istart_win=int(start_win/dt)
        iwin_width=int(win_width/dt)

        write(*,*) '*********************************'
        write(*,*) '*  WINDOWING FILE --> Welch.out  *'
```

```
          write(*,*) '**********************************'

          open(unit=10,file="welch.out",status='unknown')
          rewind 10
             call welch_window(istart_win,iwin_width,welch_amp,iadd)
             do ipk=1,iadd
               timsers(ipk)=timsers(ipk)*welch_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), welch_amp(ipk)
              endif
             enddo
          close(unit=10)
        endif


        if (i_win.eq.3) then
c............ Applying the Bartlett Window ........................
          write(*,*) 'apply Bartlett Window'
          write(*,*) 'enter the start time for the Bartlett Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Bartlett'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Bartlett Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '*************************************'
          write(*,*) '*  WINDOWING FILE --> Bartlett.out  *'
          write(*,*) '*************************************'

          open(unit=10,file="Bartlett.out",status='unknown')
          rewind 10
            call bartlett_window(istart_win,iwin_width,bart_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*bart_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), bart_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif


        if (i_win.eq.4) then
c............ Applying the Tapered Rectangular Window ................
          write(*,*) 'apply Tapered Window'
          write(*,*) 'enter the start time for the Tapered Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Tapered'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Tapered Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Tapered.out  *'
          write(*,*) '************************************'

          open(unit=10,file="Tapered.out",status='unknown')
          rewind 10
            call tapered_window(istart_win,iwin_width,tap_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*tap_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), tap_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif


        if (i_win.eq.5) then
c............ Applying the Rectangular Window ......................
          write(*,*) 'apply Tapered Window'
          write(*,*) 'enter the start time for the Rectangular Window'
          read(*,*) start_win
          write(*,*) 'enter the window width (time) for the Rectangular'
          write(*,*) 'Window'
          read(*,*) win_width
c Computing the starting and ending points of the Tapered Window
          istart_win=int(start_win/dt)
          iwin_width=int(win_width/dt)

          write(*,*) '************************************'
          write(*,*) '*  WINDOWING FILE --> Rectang.out  *'
          write(*,*) '************************************'
```

```
          open(unit=10,file="Rectang.out",status='unknown')
          rewind 10
            call rectang_window(istart_win,iwin_width,tap_amp,iadd)
            do ipk=1,iadd
              timsers(ipk)=timsers(ipk)*tap_amp(ipk)
              if (ipk.le.iadd) then
                write(10,*) tm(ipk), timsers(ipk), tap_amp(ipk)
              endif
            enddo
          close(unit=10)
        endif


c............ Padding the end of the data set ......................
        do 197 ikk=iadd+1,lxs
          timsers(ikk)=0.00D0
 197      tm(ikk)=tm(ikk-1)+dt

c............ Forward transform ...................................
          call freqf(nfs,delf,freq)
          call datax(iadd,timsers,lxs,xseries)
          call nlogn(nlog,xseries,m_sign)
          call xspect(lxs,xseries,nfs,ampt,faze)

c............ Normalizing the amplitude to account for ..............
c............ transform scaling artifacts ........................
          do 746 k=1,nfs
 746        ampt(k)=ampt(k)/float(iadd/2)

c............ Computing the energy spectrum .......................
          do 747 k=1,nfs
 747        eamp(k)=ampt(k)**2

c............ Normalizing the energy spectrum ....................
c          eampmx=eamp(1)
c          do 748 k=2,nfs
c            if(eamp(k).gt.eampmx) eampmx=eamp(k)
c 748        continue

c          do 726 k=1,nfs
c            eamp(k)=eamp(k)/eampmx
c 726      continue

c............ Normalizing the amplitude spectrum ...................
          ampmax=1.0D0
c           ampmax=ampt(1)
c           do 720 k=2,nfs
c             if (ampt(k).gt.ampmax) ampmax=ampt(k)
c 720       continue
c
          do 721 k=1,nfs
            rampt(k)=ampt(k)/ampmax
 721      continue

c............ Building the notch filter ............................
          call notch_filter(f0,sr,freq,nfs,x_notch,amp_notch,eta)

c............ Filtering the data ...................................
          do 117 i=1,nfs
            xseries(i)=x_notch(i)*xseries(i)
 117      continue

          open(unit=29,file='notch_filter.dat',status='unknown')
          rewind 29
            do 163 i=1,nfs
              rampt(i)=rampt(i)*amp_notch(i)
 163          write(29,*) freq(i), amp_notch(i), rampt(i)
          close(unit=29)

c............ Inverse Fourier Transform ............................
          m_sign=-m_sign

          call nlogn(nlog,xseries,m_sign)
          call xdata(lxs,xseries,ts)


      else if(kopt.eq.3) then

c============= Performing Convolution ================================
        write(*,*) '*****************'
        write(*,*) '*  Convolution  *'
        write(*,*) '*****************'

        write(*,*) '**********************'
        write(*,*) '*  WORK IN PROGRESS  *'
        write(*,*) '**********************'
```

```
c============= Error Message ==========================================

      else
        write(*,*) '********************'
        write(*,*) '*  INVALID OPTION  *'
        write(*,*) '********************'
        go to 814

      endif


c=====================================================================

c............ Writing to the output time-series file ................
      write(*,*) 'enter the name of the output time series'
      read(*,99) time_file

      open(unit=4,file=time_file,status='unknown')
      rewind 4
        do 113 i=1,iadd
          write(4,*) tm(i),ts(i)
 113    continue
      close(unit=4)


      stop
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine freqf(nf,df,frq)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine fills the frequency array
c     ****************************************************
c     * input nf :number of frequency values             *
c     *       df :frequency interval                     *
c     * output frq :array of computed frequency values   *
c     ****************************************************
      dimension frq(nf)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      frq(1)=0.0
      do 20 i=2,nf
        frq(i)=frq(i-1)+df
 20   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine datax(la,a,lx,x)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine converts a real time series to a
c     complex time series,with an imaginary part of zero
c     ****************************************************
c     * input  a :array containing real time series      *
c     *        la:number of values in array a            *
c     *        lx:number of values in array x            *
c     * output x :array containing complex time series   *
c     ****************************************************
      dimension x(lx),a(la)
      complex x
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      do 25 j=1,lx
        x(j)=(0.0,0.0)
 25   continue
c
      do 30 i=1,la
        x(i)=cmplx(a(i),0.0)
 30   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine spectx(nf,amp,faz,lx,x)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine creates a complex spectrum from a
c     given set of amplitudes and phases
c     ***************************************************
c     * input  amp :array of amplitude values           *
c     *        faz :array of phase values               *
c     * output  x  :complex spectrum                    *
c     ***************************************************
      dimension amp(lx),faz(lx),x(lx)
      complex x,ca,z
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      do 65 i=1,lx
        x(i)=(0.0,0.0)
```

```
 65   continue
      do 70 j=1,nf
        z=cmplx(0.0,faz(j))
        ca=cmplx(amp(j),0.0)
        x(j)=ca*cexp(z)
 70   continue
      do 71 i=1,(nf-2)
        x(i+nf)=conjg(x(nf-i))
 71   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine xdata(lx,x,a)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine computes a real time series from a
c     complex time series
c     ****************************************************
c     * input  x :complex time series                    *
c     * output a :real time series                       *
c     ****************************************************
      dimension x(lx),a(lx)
      complex x
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      do 75 i=1,lx
        a(I)=real(x(i))
 75   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine xspect(lx,x,nf,amp,faz)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine determines the amplitude and phase
c     spectra from a complex spectra
c     ****************************************************
c     * input  x :complex spectrum                       *
c     * output amp :array of amplitude values            *
c     *        faz :array of phase values                *
c     ****************************************************
      dimension x(lx),amp(nf),faz(nf)
      complex x
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      do 60 i=1,nf
        amp(i)=cabs(x(I))
        a=real(x(i))
        b=aimag(x(i))
        if (a.eq.0.0) then
          faz(I)=0.0
        else
          faz(I)=atan2(b,a)
        endif
 60   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine time(dt,lx,tme)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine computes the time values needed for
c     plotting the graphs
c     ****************************************************
c     * input  srate :sample rate                        *
c     * output  tme :time array                          *
c     ****************************************************
      dimension tme(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      tme(1)=0.0
      do 80 i=2,lx
      tme(i)=tme(i-1)+dt
 80   continue
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine nlogn(nlog,x,m_sign)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this is a fft program
c     ****************************************************
c     * input  m_sign :-1 for complex spectrum           *
c     *               +1 for complex time series         *
c     *        nlog :lx=2**nlog                          *
c     *          x  :complex series                      *
```

```
c     * output  x :output complex series                  *
c     ****************************************************
      dimension x(*),m(25)
      complex x,wk,hold,q
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      n=nlog
      lx=2**n
      do 1 i=1,n
        m(i)=2**(n-i)
 1    continue

      do 4 l=1,n
        nblock=2**(l-1)
        lblock=lx/nblock
        lbhalf=lblock/2
        k=0

        do 4 iblock=1,nblock
          fk=k
          flx=lx
          pi2=8.0*atan(1.0)
          v=m_sign*pi2*fk/flx
          wk=cmplx(cos(v),sin(v))
          istart=lblock*(iblock-1)

            do 2 i=1,lbhalf
              j=istart+i
              jh=j+lbhalf
              q=x(jh)*wk
              x(jh)=x(j)-q
              x(j)=x(j)+q
 2          continue

            do 3 i=2,n
              ii=i
              if (k.lt.m(i)) go to 4
              k=k-m(i)
 3          continue

 4    k=k+m(ii)

      k=0
      do 7 j=1,lx
        if (k.lt.j) go to 5
        hold=x(j)
        x(j)=x(k+1)
        x(k+1)=hold

 5      do 6 i=1,n
          ii=i
          if (k.lt.m(i)) go to 7
          k=k-m(i)
 6      continue

 7    k=k+m(ii)

      if (m_sign.lt.0) return

      do 8 i=1,lx
        x(i)=x(i)/flx
 8    continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine hamming (la1,la2,amp,lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     computes the discrete values for the Hamming Window
c     ******************************************************
c     * input la1: the number zeros preceeding the window    *
c     *       la2: width of the window                       *
c     *        lx: array dimension                           *
c     *                                                      *
c     ******************************************************
      dimension amp(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      pi=4.0*atan(1.0)
      do i=1,lx
        if ((i.le.la1).or.(i.gt.(la2+la1))) then
          amp(i)=0.0
        else
          amp(i)=0.54-0.46*cos(2*pi*(i-(la1+1))/(la2-1))
        endif
      enddo
      return
```

```
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      Subroutine Welch_window (la1,la2,amp,lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Computes the coefficients of the Welch Window.                        c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      dimension amp(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      bottom=0.5*la2

      do i=1,lx
        if ((i.le.la1).or.(i.gt.(la2+la1))) then
          amp(i)=0.0
        else
          top=(i-(la1+1))-bottom
          temp=top/bottom
          temp2=temp*temp
          amp(i)=1-temp2
        endif
      end do

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine Bartlett_window(la1,la2,amp,lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Computes the coefficients of the Bartlett Window.                     c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      dimension amp(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      bottom=0.5*la2

      do i=1,lx
        if ((i.le.la1).or.(i.gt.(la2+la1))) then
          amp(i)=0.0
        else
          top=(i-(la1+1))-bottom
          temp=top/bottom
          amp(i)=1-abs(temp)
        endif
      end do

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine Tapered_window(la1,la2,amp,lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Computes the coefficients of the Tapered Window.                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      parameter (m=40)
      dimension amp(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      pi=4.0*atan(1.0)
      l=la2-(2*m)

      do 10 j=1,lx
        if (((j-1).le.la1).or.((j-1).ge.(la2+la1))) then
          amp(j)=0.0
        else
          if (((j-1).lt.(la1+m)).and.((j-1).gt.la1)) then
            amp(j)=(1.0+cos(pi*((j-1)+(la1+m))/m))/2.0
          else if (((j-1).gt.(la1+m+1)).and.((j-1).lt.(la1+la2))) then
            amp(j)=(1.0+cos(pi*((j-1)-(la1+m+1))/m))/2.0
          else
            amp(j)=1.0
          endif
        endif
 10   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine rectang_window(la1,la2,amp,lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Computes the coefficients of the Rectangular Window.                  c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      parameter (m=20)
      dimension amp(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      pi=4.0*atan(1.0)
```

```
      l=la2-(2*m)

      do 10 j=1,lx
        if (((j-1).le.la1).or.((j-1).ge.(la2+la1))) then
          amp(j)=0.0
        else
          amp(j)=1.0
        endif
 10   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine bfilter(fcut,lx,num,y,n,delf)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     this subroutine generates both the low and high pass
c     butterworth filters
c     *******************************************************
c     * input  fcut :cutoff frequency                       *
c     *        lx :number of points in the filter           *
c     *        num :tells which filter is to be generated   *
c     *             =1 low pass                             *
c     *             =2 high pass                            *
c     *        delf :frequency increment                    *
c     *        n :gives the order of the filter             *
c     * output y :stores the filter generated               *
c     *******************************************************
      dimension y(lx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      if (num.eq.1) then
        do 200 i=1,lx
          y(i)=1.0/(1.0+(((i*delf)/fcut)**(2.0*n)))
 200    continue
      else
        do 201 j=1,lx
          y(j)=1.0/(1.0+((fcut/(j*delf))**(2.0*n)))
 201    continue
      endif
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine convl(data,n,respns,m,isign,ans)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c Performs CONVOLUTION and DECONVOLUTION. Taken from Numerical        c
c Recipes (2nd Edition). Response function stored in wrap-around for- c
c mat.                                                                c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      parameter (nmax=80000)

      complex ans(n), cfft(nmax), x_data(nmax), x_respns(nmax)
      dimension respns(n)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c Putting respns in array of length n
      do 11 i=1,(m-1)/2
        respns(n+1-i)=respns(m+1-i)
 11   continue

c Padding with zeros
      do 12 i=(m+3)/2,n-(m-1)/2
        respns(i)=0.0
 12   continue

c Perform FFT on both time-series
      call datax(n,data,n,x_data)
      call nlogn(nlog,x_data,sign)

      call datax(n,respns,n,x_respns)
      call nlogn(nlog,x_respns,sign)

      no2=n/2
c
c Performing convolution and deconvolution
      do 13 i=1,(no2+1)
        if(isign.eq.1) then
          ans(i)=x_data(i)*x_respns(i)/no2
        else if (isign.eq.-1) then
          if(cabs(x_respns(i)).eq.0.0) pause 'deconvolving at response
     # zero in convl'
            x_respns(i)=x_data(i)/x_respns(i)/no2
        else
          pause 'no meaning for isign in convl'
        endif
 13   continue
```

```
      x_respns(1)=cmplx(real(x_respns(1)),real(ans(no2+1)))
      call xdata(n,x_respns,ts)
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine Notch_filter(f0,sr,frq,nf,x_notch,amp_notch,eta)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c This subroutine computes the frequency response of a Notch filter.  c
c Approach taken from Numerical Recipes.                              c
c                                                                     c
c eta: control parameter for the notch filter.                        c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      complex x_temp, x_temp2, x_notch
      dimension frq(nf), x_notch(nf)
      dimension amp_notch(nf)

c      parameter (eta=0.00001)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      pi=4.0*atan(1.0)

c Conver the frequency to be removed to W-space.
      w0=tan(pi*sr*f0)

      do 10 i=1,nf
        w=tan(pi*frq(i)*sr)
        w2=w*w
        w02=w0*w0
        w3=eta*w0
        x_temp=cmplx(w,-w3)
        x_temp2=x_temp*x_temp
        x_notch(i)=(w2-w02)/(x_temp2-w02)
        amp_notch(i)=cabs(x_notch(i))
 10   continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine interpolate(x,y,npt,npts,dx)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Interpolates using polynomial approximations                        c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      character*20  infile, outfile
      character*80  title
      dimension x(80000),y(80000),xp(80000),yp(80000),ypn2(80000)

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      pi=4.0D0*datan(1.0D0)

      write(*,*) '*****************************************************'
      write(*,*) '   NUMBER OF POINTS IN INPUT DATA ... ', npt
      write(*,*) '*****************************************************'
      write(*,*) ' '

      write(*,*) 'OPTIONS'
      write(*,*) '1.     32 point approximation'
      write(*,*) '2.     64 point approximation'
      write(*,*) '3.    128 point approximation'
      write(*,*) '4.    256 point approximation'
      write(*,*) '5.    512 point approximation'
      write(*,*) '6.   1024 point approximation'
      write(*,*) '7.   2048 point approximation'
      write(*,*) '8.   4096 point approximation'
      write(*,*) '9.   8192 point approximation'
      write(*,*) '10. 16384 point approximation'
      write(*,*) 'enter option'
      read(*,*) nopt

c  Choose interpolation methodology
      write(*,*) 'Select interpolation procedure'
      write(*,*) '1 ... Linear Interpolation'
      write(*,*) '2 ... Natural Spline'
      read(*,*) intpol

      if(nopt.eq.1) then
       npts=32
       dx=(x(npt)-x(1))/float(npts-1)
      else
       if(nopt.eq.2) then
        npts=64
        dx=(x(npt)-x(1))/float(npts-1)
       else
        if (nopt.eq.3) then
```

```
          npts=128
          dx=(x(npt)-x(1))/float(npts-1)
         else
          if (nopt.eq.4) then
           npts=256
           dx=(x(npt)-x(1))/float(npts-1)
          else
           if (nopt.eq.5) then
            npts=512
            dx=(x(npt)-x(1))/float(npts-1)
           else
            if (nopt.eq.6) then
             npts=1024
             dx=(x(npt)-x(1))/float(npts-1)
            else
             if (nopt.eq.7) then
              npts=2048
              dx=(x(npt)-x(1))/float(npts-1)
             else
              if (nopt.eq.8) then
               npts=4096
               dx=(x(npt)-x(1))/float(npts-1)
              else
               if (nopt.eq.9) then
                npts=8192
                dx=(x(npt)-x(1))/float(npts-1)
               else
                if (nopt.eq.10) then
                 npts=16384
                 dx=(x(npt)-x(1))/float(npts-1)
                else
                 write(*,*) 'INPUT ERROR'
                 go to 901
                endif
               endif
              endif
             endif
            endif
           endif
          endif
         endif
        endif
       endif
      endif

c  XP(i) refers to the interpolation locations

      do 30 i=1,npts
       xp(i)=x(1)+(i-1)*dx
 30   continue

c  Boundary conditions
      ypp1=0.00D0
      yppn=0.00D0

c  Performing Linear Interpolation
      if (intpol.eq.1) then
       do 40 i=2,npts-1
        do 45 j=1,npt
         if(x(j).ge.xp(i)) then
          call intpl(x(j-1),x(j),y(j-1),y(j),xp(i),yp(i))
          go to 40
         endif
 45      continue
 40     continue
      else
       call spline(x,y,npt,ypp1,yppn,ypn2)
       do 80 i=2,npts-1
         call splint(x,y,ypn2,npt,xp(i),yp(i))
 80     continue
      endif

c  Assigning end points
      yp(npts)=y(npt)
      yp(1)=y(1)

c Over-writing the input and output files
      if (npts.gt.npt) then
        nc=npts
      else
        nc=npt
      endif

      do 150 j=1,nc
        x(j)=0.0
        y(j)=0.0
 150  continue

      do 155 j=1,npts
```

```
        y(j)=yp(j)
        x(j)=xp(j)
 155  continue

 901  continue

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine intpl(x1,x2,y1,y2,x,y)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Reads in the coordinates of the 2 closest points and uses the    c
c  passed x value to calculate the required y value.                c
c  LINEAR INTERPOLATION                                             c
c  y=mx+c                                                           c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      sp=(y2-y1)/(x2-x1)
      c=y2-(sp*x2)
      y=(sp*x)+c

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SPLINE(X,Y,N,YP1,YPN,Y2)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  This subroutine performs spline interpolation. Based on routines c
c  from Numerical Recipes.                                          c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      PARAMETER (NMAX=80000)
      DIMENSION X(N),Y(N),Y2(N),U(NMAX)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      IF (YP1.GT..99E30) THEN
        Y2(1)=0.
        U(1)=0.
      ELSE
        Y2(1)=-0.5
        U(1)=(3./(X(2)-X(1)))*((Y(2)-Y(1))/(X(2)-X(1))-YP1)
      ENDIF
      DO 11 I=2,N-1
        SIG=(X(I)-X(I-1))/(X(I+1)-X(I-1))
        P=SIG*Y2(I-1)+2.
        Y2(I)=(SIG-1.)/P
        U(I)=(6.*((Y(I+1)-Y(I))/(X(I+1)-X(I))-(Y(I)-Y(I-1))
     *      /(X(I)-X(I-1)))/(X(I+1)-X(I-1))-SIG*U(I-1))/P
 11   CONTINUE
      IF (YPN.GT..99E30) THEN
        QN=0.
        UN=0.
      ELSE
        QN=0.5
        UN=(3./(X(N)-X(N-1)))*(YPN-(Y(N)-Y(N-1))/(X(N)-X(N-1)))
      ENDIF
      Y2(N)=(UN-QN*U(N-1))/(QN*Y2(N-1)+1.)
      DO 12 K=N-1,1,-1
        Y2(K)=Y2(K)*Y2(K+1)+U(K)
 12   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SPLINT(XA,YA,Y2A,N,X,Y)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  Performs the actual interpolation based on the 2nd derivatives   c
c  produced by SPLINE. Result is a cubic spine interpolation.       c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      DIMENSION XA(N),YA(N),Y2A(N)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      KLO=1
      KHI=N
 1    IF (KHI-KLO.GT.1) THEN
        K=(KHI+KLO)/2
        IF(XA(K).GT.X)THEN
          KHI=K
        ELSE
          KLO=K
        ENDIF
      GOTO 1
      ENDIF
      H=XA(KHI)-XA(KLO)
      IF (H.EQ.0.) PAUSE 'Bad XA input.'
      A=(XA(KHI)-X)/H
```

```
      B=(X-XA(KLO))/H
      Y=A*YA(KLO)+B*YA(KHI)+
     *      ((A**3-A)*Y2A(KLO)+(B**3-B[illegible]A(KHI))*(H**2)/6.

      RETURN
      END
```